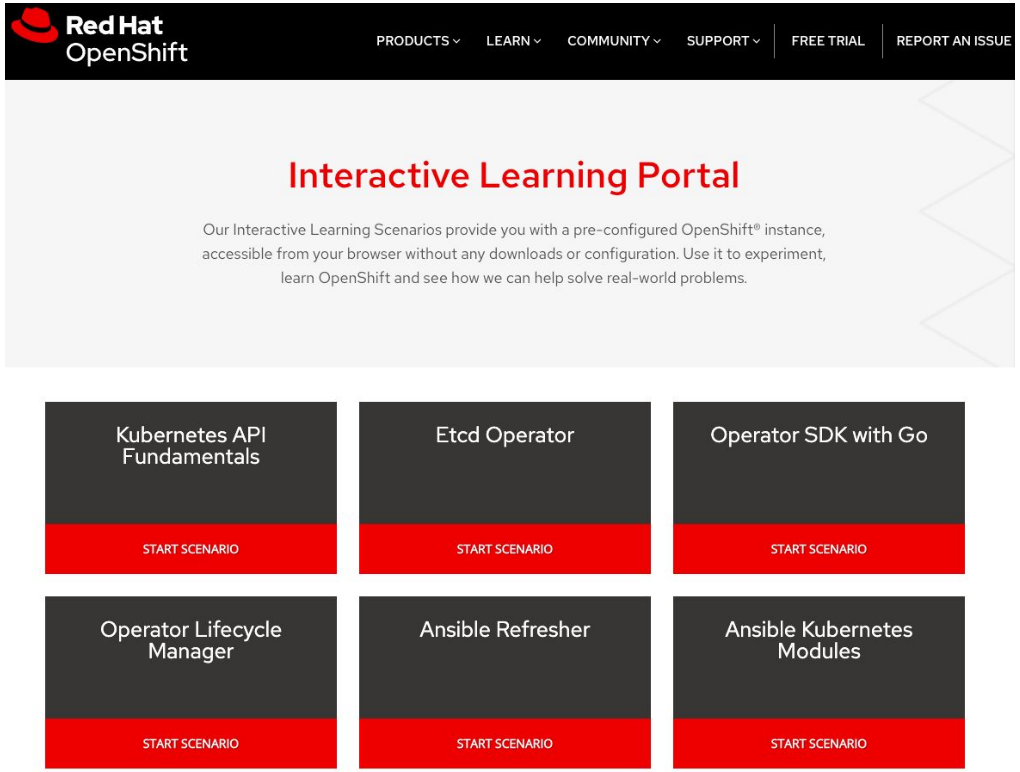
Red Hat
OpenShift

Introducing Operator-SDK 1.0.0

Part of the Operator Framework - a toolkit to manage kubernetes native applications in an effective, automated, scalable way.

Brought to you by the Red Hat Operator Enablement Team

learn.openshift.com/operatorframework



Interactive Learning Portal

Our Interactive Learning Scenarios provide you with a pre-configured OpenShift® instance, accessible from your browser without any downloads or configuration. Use it to experiment, learn OpenShift and see how we can help solve real-world problems.

Kubernetes API Fundamentals

Etcd Operator

Operator SDK with Go

Operator Lifecycle Manager

Ansible Refresher

Ansible Kubernetes Modules

HERE TO HELP YOU **SUCCEED** WITH BATTLE-TESTED TOOLS.

OPERATOR FRAMEWORK

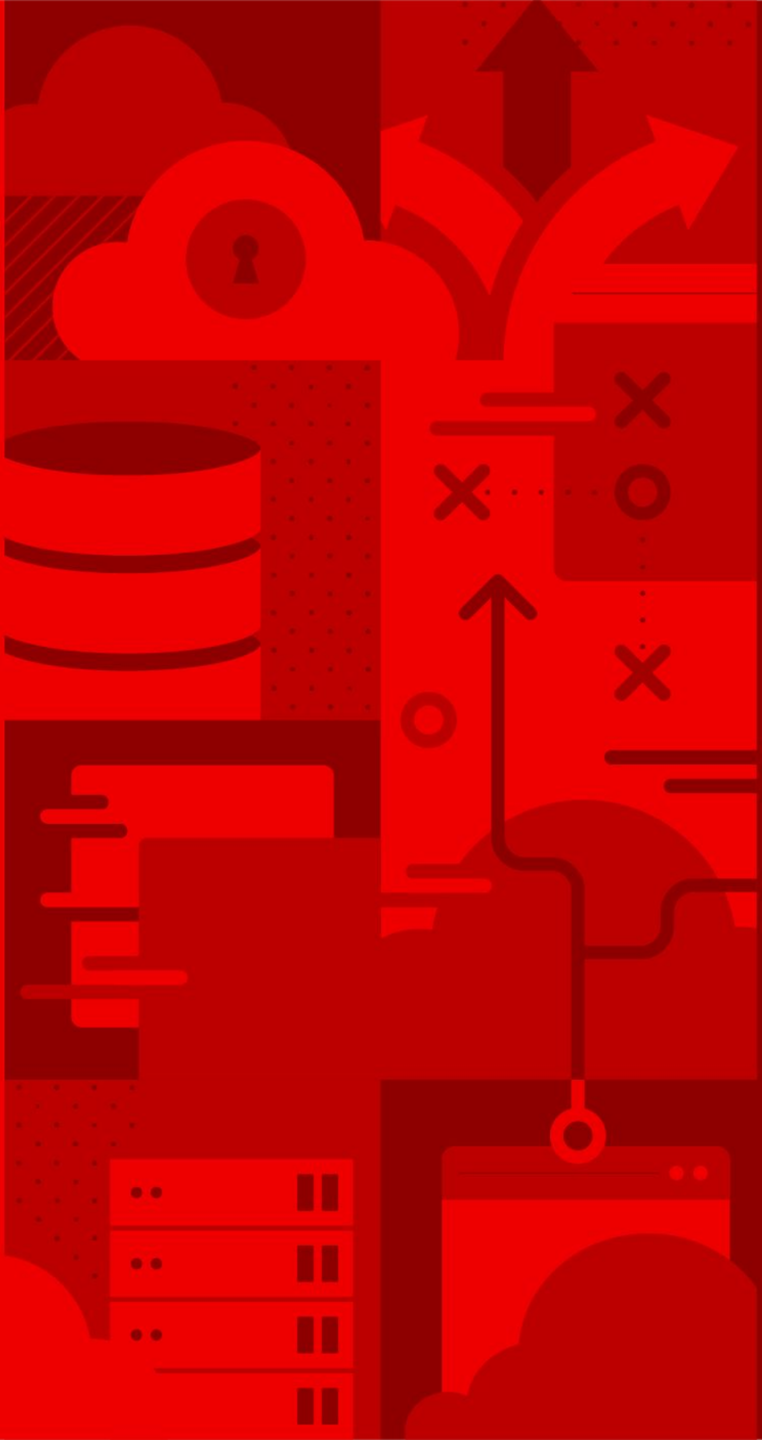
Sources:
https://github.com/operator-framework

OPERATOR SDK = Building/Dev

OPERATOR LIFECYCLE MANAGER = Install/Manage

Sources:
https://github.com/operator-framework/operator-sdk
https://github.com/operator-framework/operator-lifecycle-manager

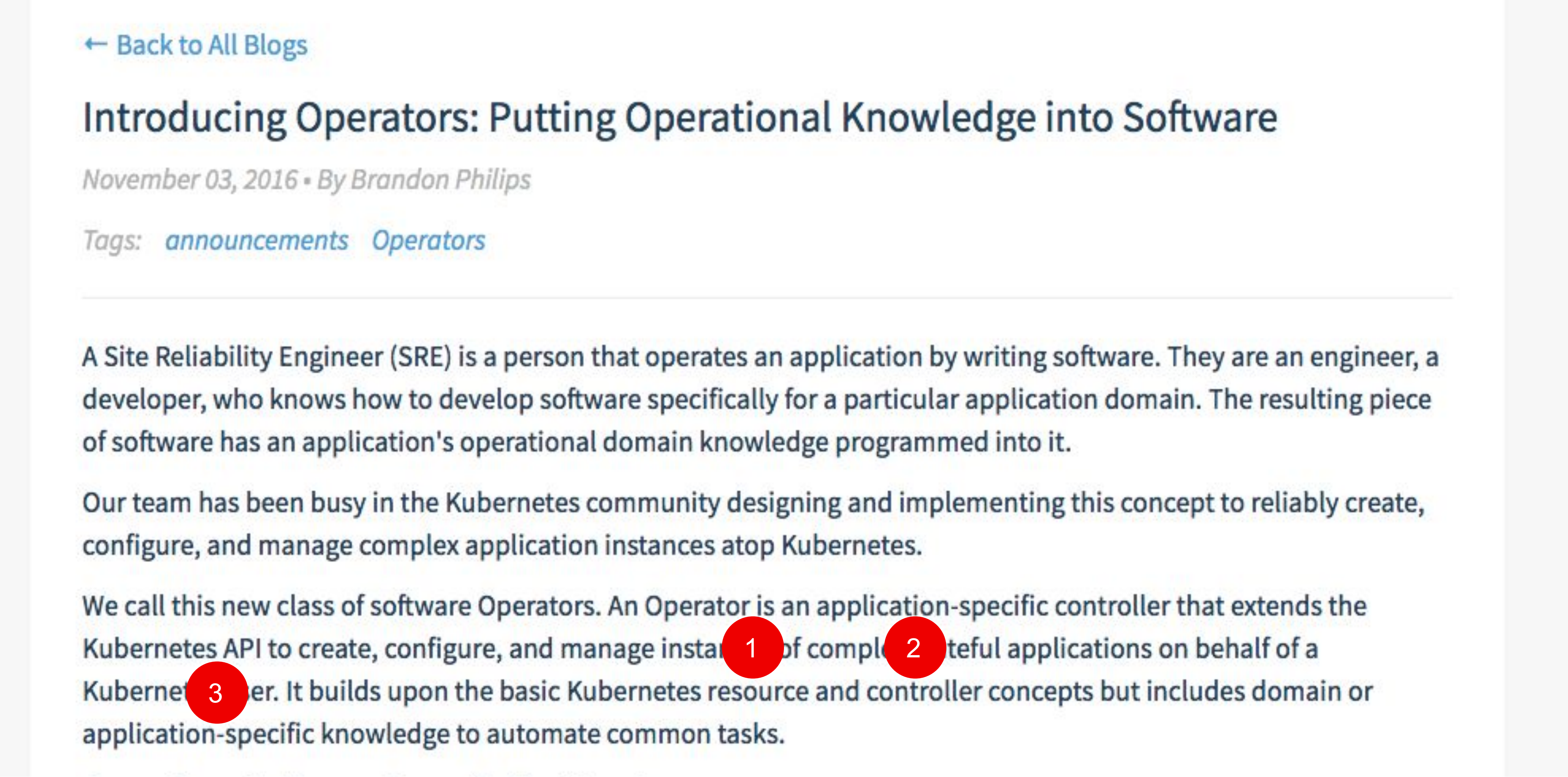
WHAT IS AN **OPERATOR**?



Operators

An operator represents human operational knowledge in software, to reliably manage an application.

What is an Operator



← Back to All Blogs

Introducing Operators: Putting Operational Knowledge into Software

November 03, 2016 • By Brandon Philips

Tags: announcements Operators

A Site Reliability Engineer (SRE) is a person that operates an application by writing software. They are an engineer, a developer, who knows how to develop software specifically for a particular application domain. The resulting piece of software has an application's operational domain knowledge programmed into it.

Our team has been busy in the Kubernetes community designing and implementing this concept to reliably create, configure, and manage complex application instances atop Kubernetes.

We call this new class of software Operators. An Operator is an application-specific controller that extends the Kubernetes API to create, configure, and manage insta 1 of compl 2 teful applications on behalf of a Kubernet 3 er. It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

Source:
https://coreos.com/blog/introducing-operators.html

What is an Operator

It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

1

Resource

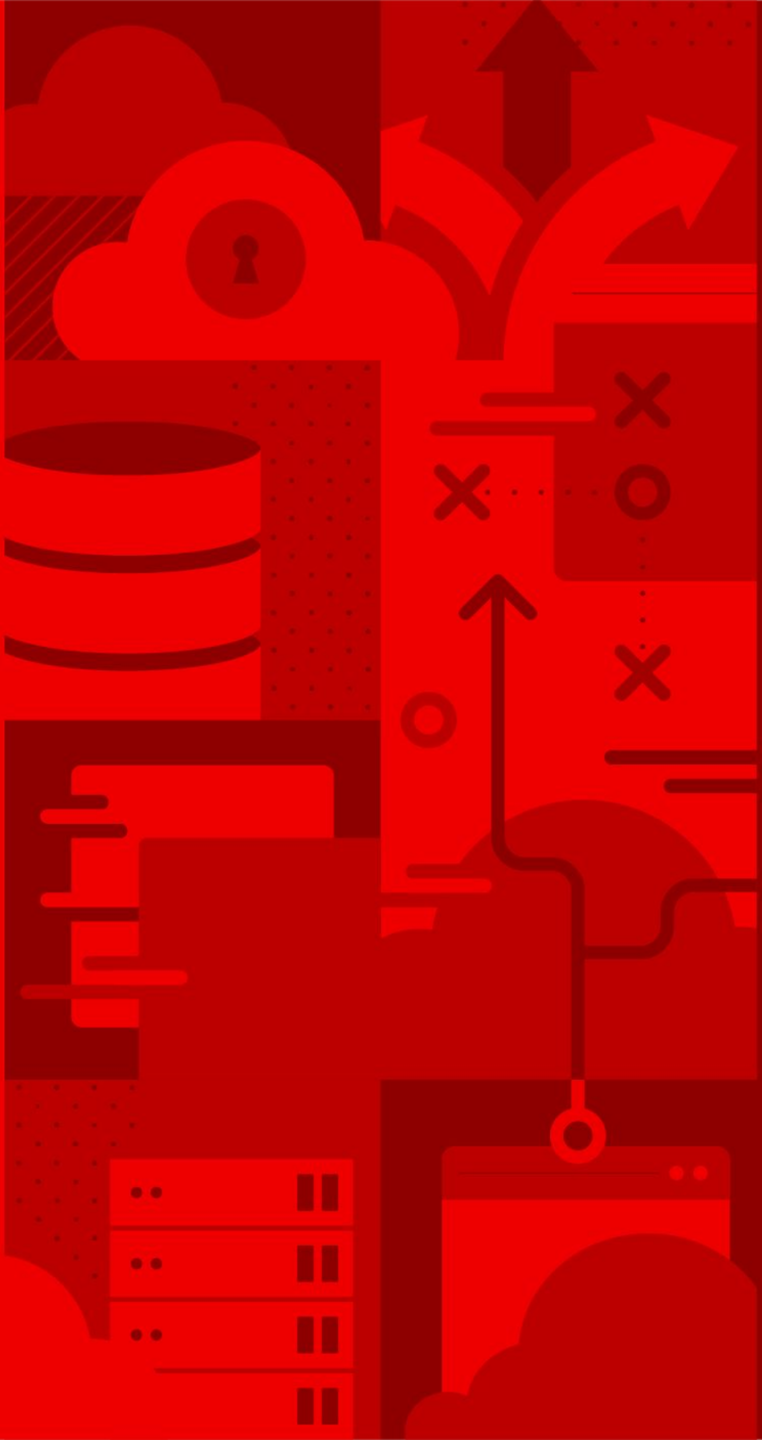
2

Controller

3

Knowledge

Source:
https://coreos.com/blog/introducing-operators.html



Resource

an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind

Pod

the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents processes running on your Cluster.

Source:
https://kubernetes.io/docs/concepts/workloads/pods/pod/

ConfigMap

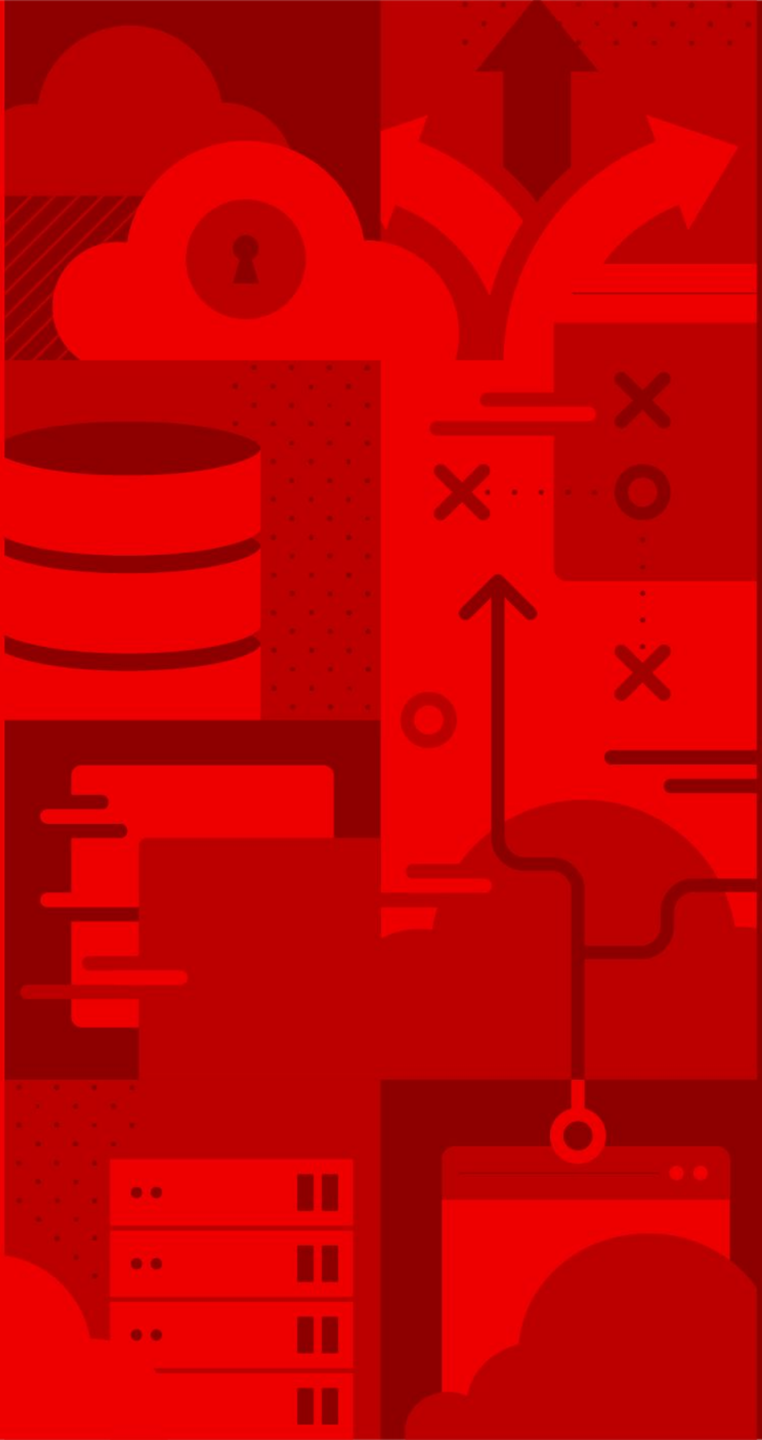
provides a way to inject configuration data into Pods. The data stored in a ConfigMap object can be referenced in a volume of type configMap and then consumed by containerized applications running in a Pod.

Source:
https://kubernetes.io/docs/concepts/storage/volumes/#configmap

Route (Ingress)

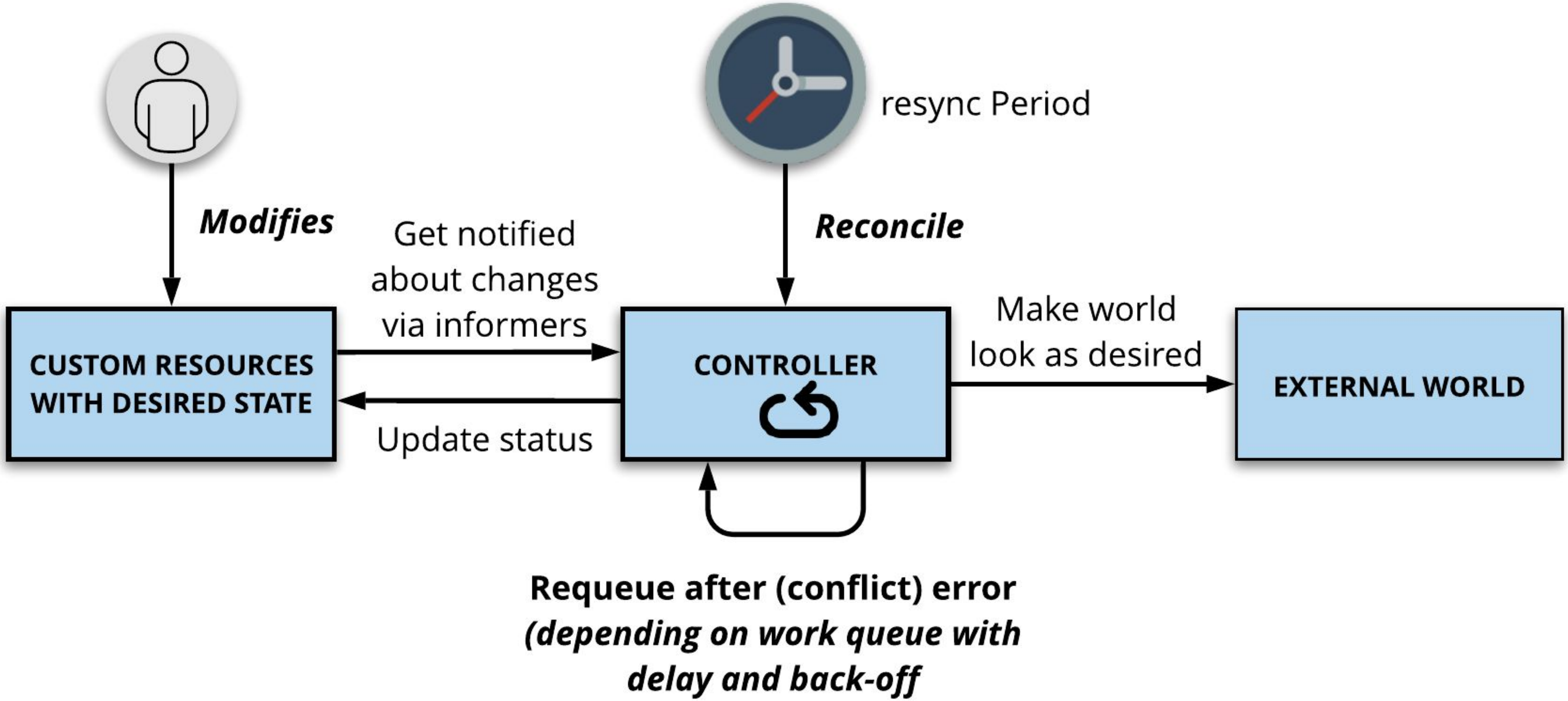
a way to expose a service by giving it an externally-reachable hostname like www.example.com.

Source:
https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/routes.html



Controller

control loop that watches the state of your cluster and moves the current cluster state closer to the desired state


Modifies
Get notified about changes via informers
resync Period
Reconcile
CUSTOM RESOURCES WITH DESIRED STATE
Update status
CONTROLLER
Make world look as desired
EXTERNAL WORLD
Requeue after (conflict) error
(depending on work queue with delay and back-off

ReplicaSet Controller

defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.

Source:
https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Deployment Controller

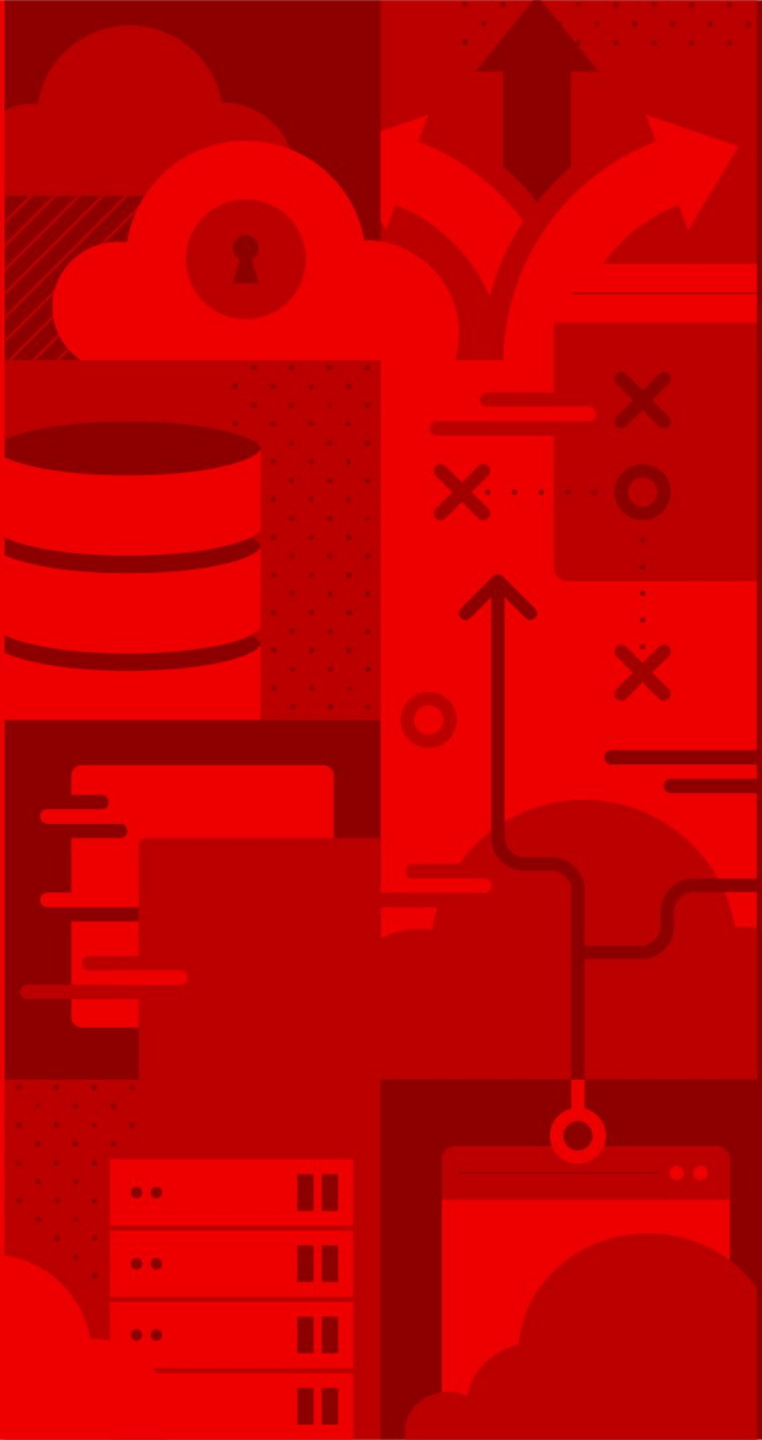
provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.

Source:
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

DaemonSet Controller

ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.

Source:
https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/



Knowledge

domain or application specific; usually must be learned from users and/or administrators rather than developers

Domain or Application Specific Knowledge

real-world experience with managing your application(s)

Install

Self Heal

Scale

Update

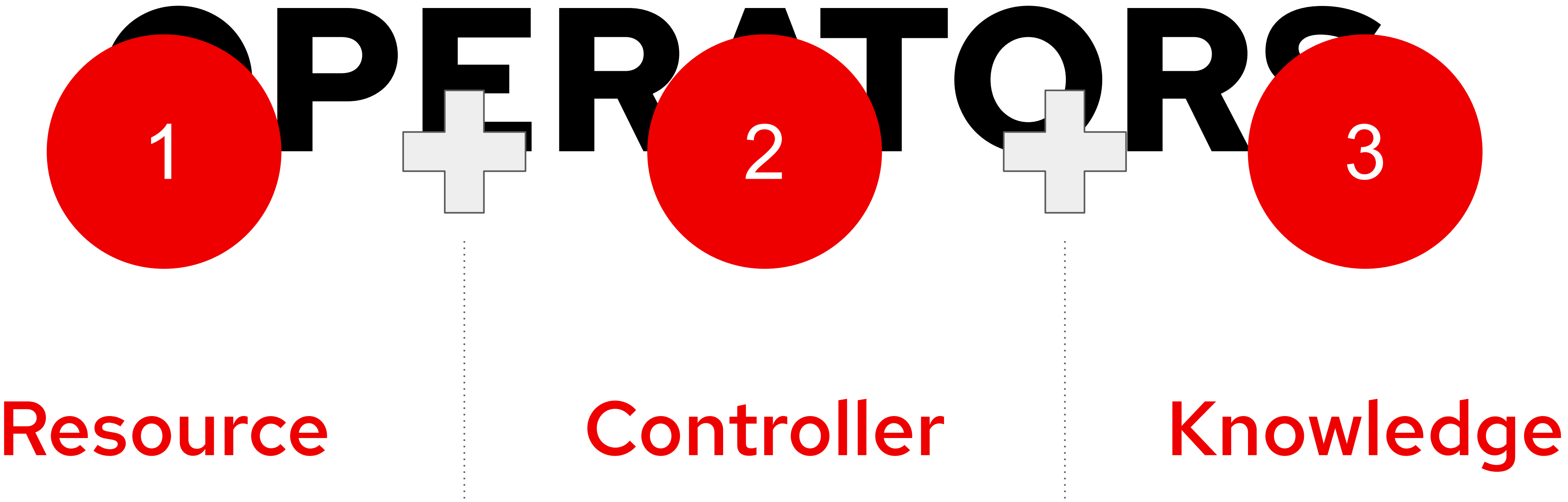
Backup

Clean Up

Observability

Resiliency

It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.



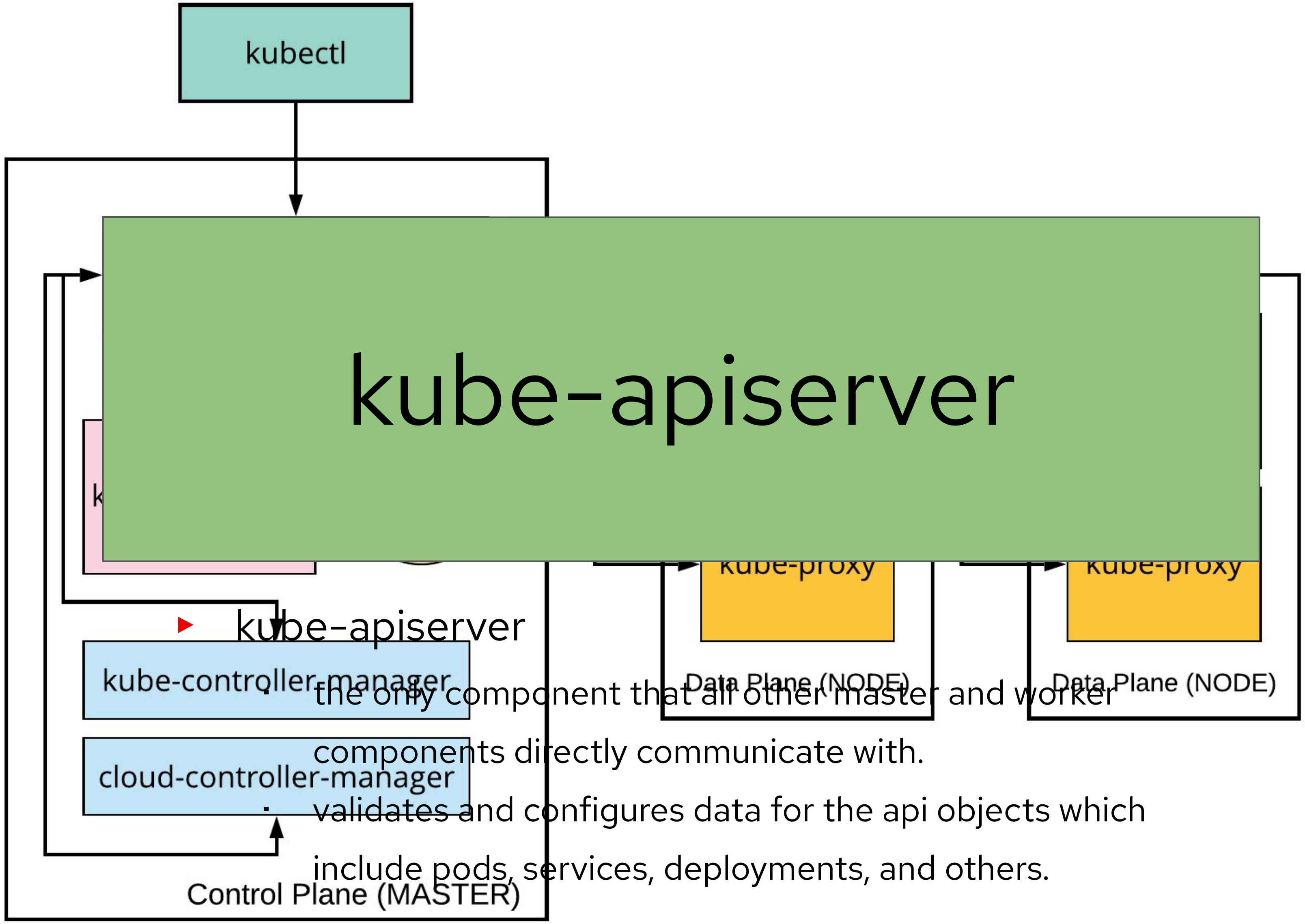
Resource

Controller

Knowledge

Source:
https://coreos.com/blog/introducing-operators.html

An Operator takes advantage of what Kubernetes does best



- kube-apiserver

the only component that all other master and worker components directly communicate with.

validates and configures data for the api objects which include pods, services, deployments, and others.

Source:
https://medium.com/@karthikeyan_krishnaswamy/overview-of-kubernetes-34d8e0e59b26

```
curl –s localhost:8001/api/v1 | jq –r .resources[].name

bindings
componentstatuses
configmaps
endpoints
events
limitranges
namespaces
namespaces/finalize
namespaces/status
nodes
...
```

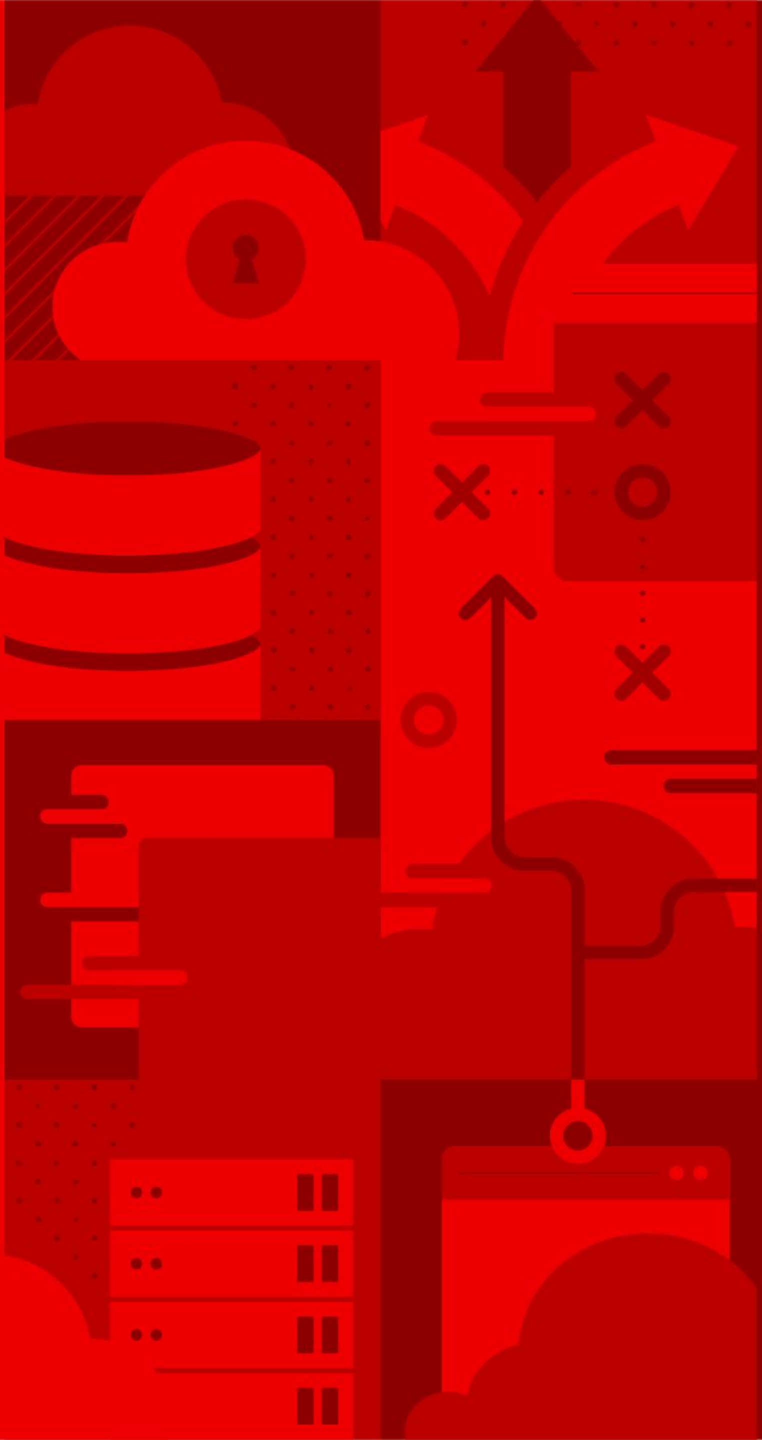
```
redhat:mhillsma deploy $ oc get -n openshift-dns pods
NAME                READY   STATUS    RESTARTS   AGE
...
dns-default-vxvth   3/3     Running   0          5d8h
```

```
(curl -s -XGET localhost:8001/api/v1/namespaces/openshift-dns/pods | jq -r
.items[].metadata.name)

"dns-default-478pn"
"dns-default-4fv5s"
"dns-default-vxvth"
"dns-default-7k289"
"dns-default-fw7gv"
"dns-default-j7mzv"
```

```
redhat:mhillsma deploy $ oc get -n openshift-dns pod/dns-default-vxvth -o yaml
apiVersion: v1
kind: Pod
metadata:
...
  name: dns-default-vxvth

(curl -XGET localhost:8001/api/v1/namespaces/openshift-dns/pods/dns-default-vxvth)

apiVersion: v1
kind: Pod
metadata:
  name: dns-default-vxvth
  namespace: openshift-dns
  ownerReferences:
...
```

CRDs allow us to **EXTEND** the Kubernetes API

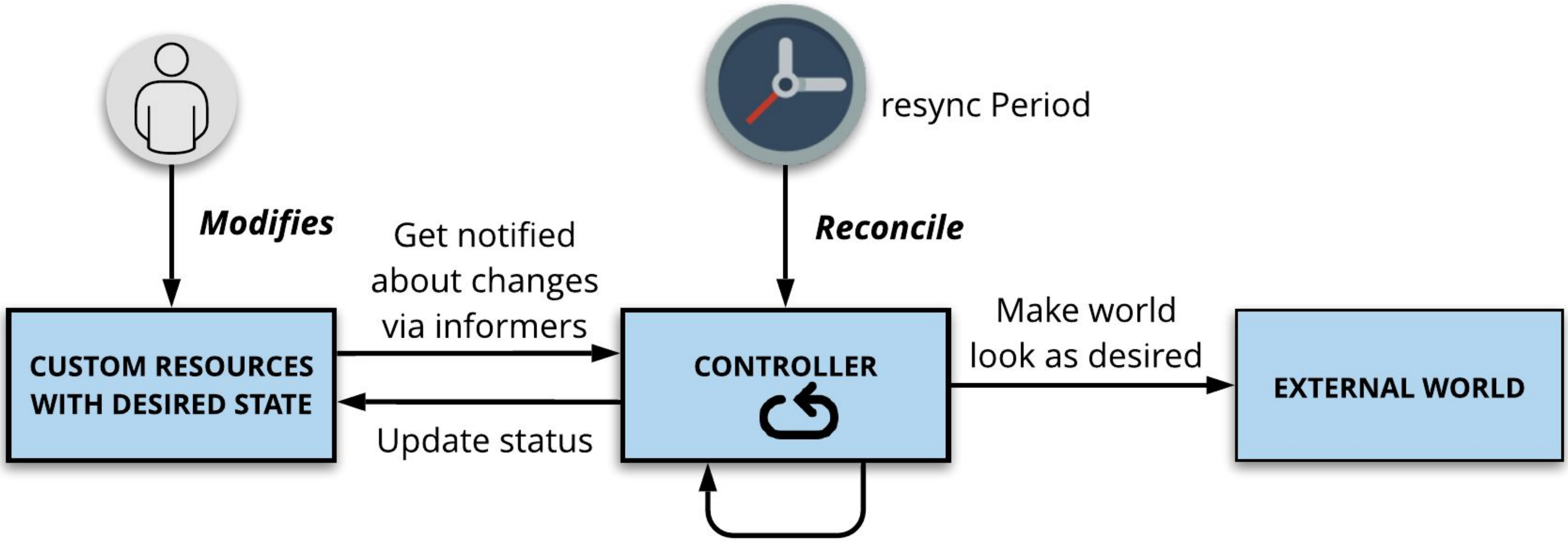
- modify the API without recompiling
- create our very own API resource/object
- resource/object exists but nothing acts on its presence and this is where controllers come in



A Custom Resource <u>needs</u> a controller to **ACT** upon its presence.

What do we mean by **ACT**?

- Create
- Read
- Update
- Delete



Modifies

Get notified about changes via informers

Update status

resync Period

Reconcile

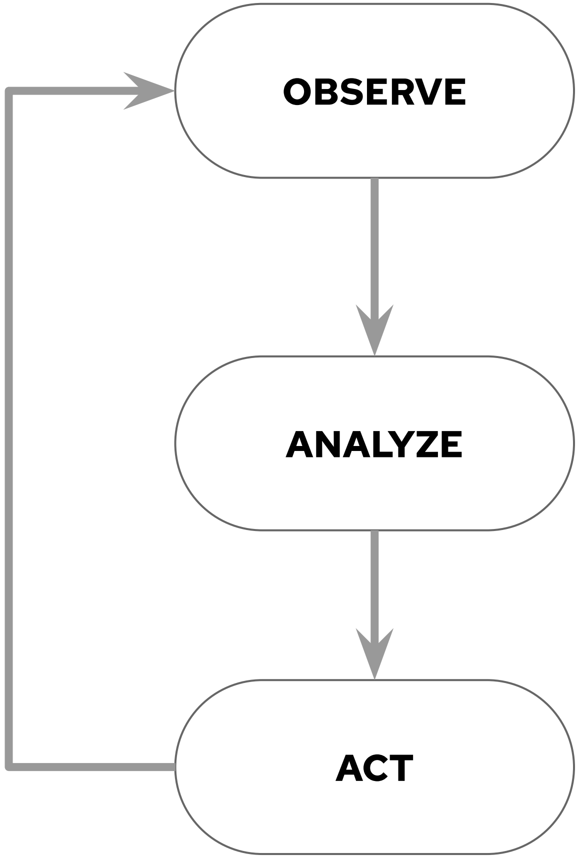
Make world look as desired

CUSTOM RESOURCES WITH DESIRED STATE

CONTROLLER

EXTERNAL WORLD

Requeue after (conflict) error
(depending on work queue with delay and back-off



OBSERVE

Current state of the cluster.

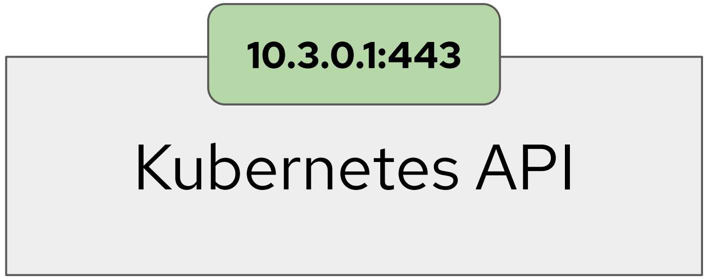
ANALYZE

Compare current state to desired state.

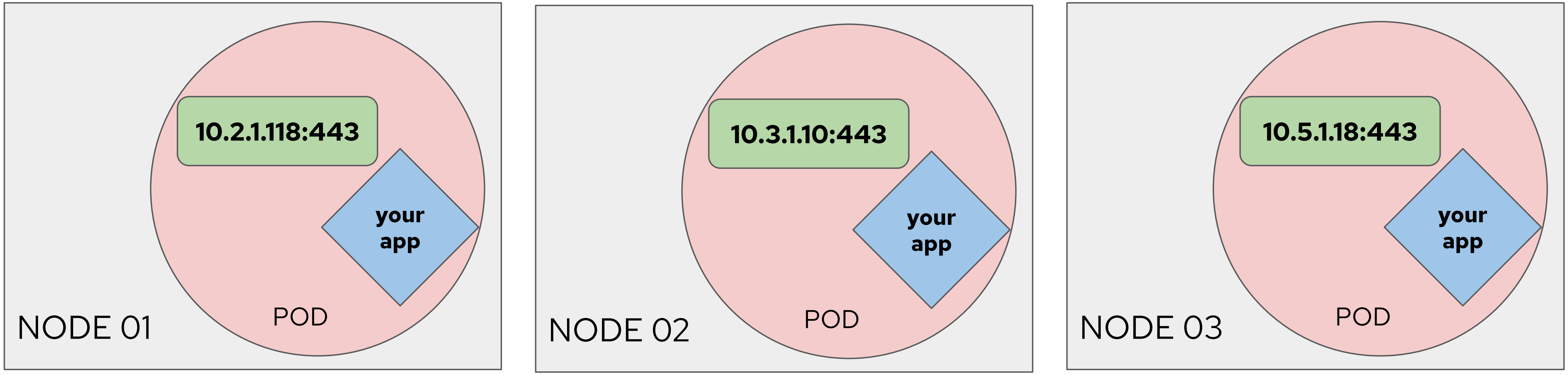
ACT

Perform all the actions necessary to make current state meet desired state.

What is an Operator

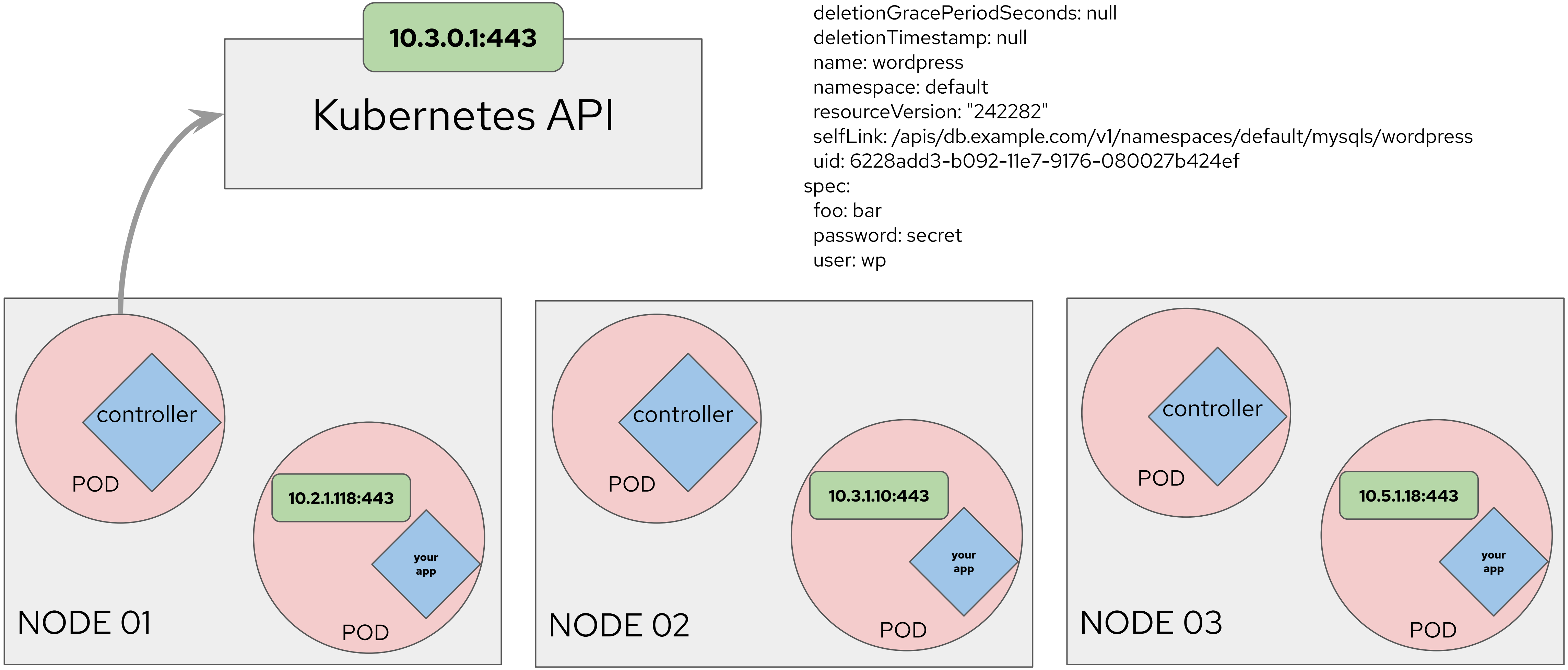


```
apiVersion: db.example.com/v1
kind: MySql
metadata:
  clusterName: ""
  creationTimestamp: 2017-10-14T03:47:21Z
  deletionGracePeriodSeconds: null
  deletionTimestamp: null
  name: wordpress
  namespace: default
  resourceVersion: "242282"
  selfLink: /apis/db.example.com/v1/namespaces/default/mysqls/wordpress
  uid: 6228add3-b092-11e7-9176-080027b424ef
spec:
  foo: bar
  password: secret
  user: wp
```



What is an Operator

```
apiVersion: db.example.com/v1
kind: MySql
metadata:
  clusterName: ""
  creationTimestamp: 2017-10-14T03:47:21Z
  deletionGracePeriodSeconds: null
  deletionTimestamp: null
  name: wordpress
  namespace: default
  resourceVersion: "242282"
  selfLink: /apis/db.example.com/v1/namespaces/default/mysqls/wordpress
  uid: 6228add3-b092-11e7-9176-080027b424ef
spec:
  foo: bar
  password: secret
  user: wp
```



What do we mean by **ACT**?

- Create
- Read
- Update
- Delete

Create, Read, Update, Delete...Probably Not Enough

Server startup/shutdown
Mastering the mysqladmin administrative client
Using the mysql interactive client
User account maintenance
Log file maintenance
Database backup/copying
Hardware tuning
Multiple server setups
Software updates and upgrades
File system security
Server security
Repair and maintenance
Crash recovery
Preventive maintenance
Understanding the mysqld server daemon
Performance analysis
Choosing what else to install (e.g. Apache, Perl +modules, PHP)
Which version of MySQL (stable, developer, source, binary)
Creating a user acccount for the mysql user and group
Download and unpack a distribution
Compile source code and install (or rpm)
Initialize the data directory and grant tables with mysql_install_db
Starting the server
Installing Perl DBI support
Installing PHP
Installing Apache
Obtaining and installing the samp_db sample database

Securing a new MySQL installation
Running mysqld as an unprivileged user
Methods of starting the server
Invoking mysqld directly
Invoking safe_mysqld
Invoking mysql.server
Specifying startup options
Checking tables at startup
Shutting down the server
Regaining control of the server if you can't connect
Creating new users and granting privileges
Determining who can connect from where
Who should have what privileges?
Administrator privileges
Revoking privileges
Removing users
deciding/finding the Data Directory's location
Structure of the Data Directory
How mysqld provides access to data
Running multiple servers on a single Data Directory
Database representation
Table representation (form, data and index files)
OS constraints on DB and table names
Data Directory structure and performance, resources, security
MySQL status files (.pid, .err, .log, etc)
Relocating Data Directory contents

Creating new users and granting privileges
Determining who can connect from where
Who should have what privileges?
Administrator privileges
Revoking privileges
Removing users
Methods: mysqldump vs. direct copying
Backup policies
Scheduled cycles
Update logging
Consistent and comprehensible file-naming
Backing up the backup files
Off-site / off-system backups
Backing up an entire database with mysqldump
Compressed backup files
Backing up individual tables
Using mysqldump to transfer databases to another server
mysqldump options (flush-logs, lock-tables, quick, opt)
Direct copying methods
Database replication (live and off-line copying)
Recovering an entire database
Recovering grant tables
Recovering from mysqldump vs. tar/cpio files
Using update logs to replay post-backup queries
Editing update logs to avoid replaying erroneous queries
Recovering individual tables
Default parameters

What is an Operator

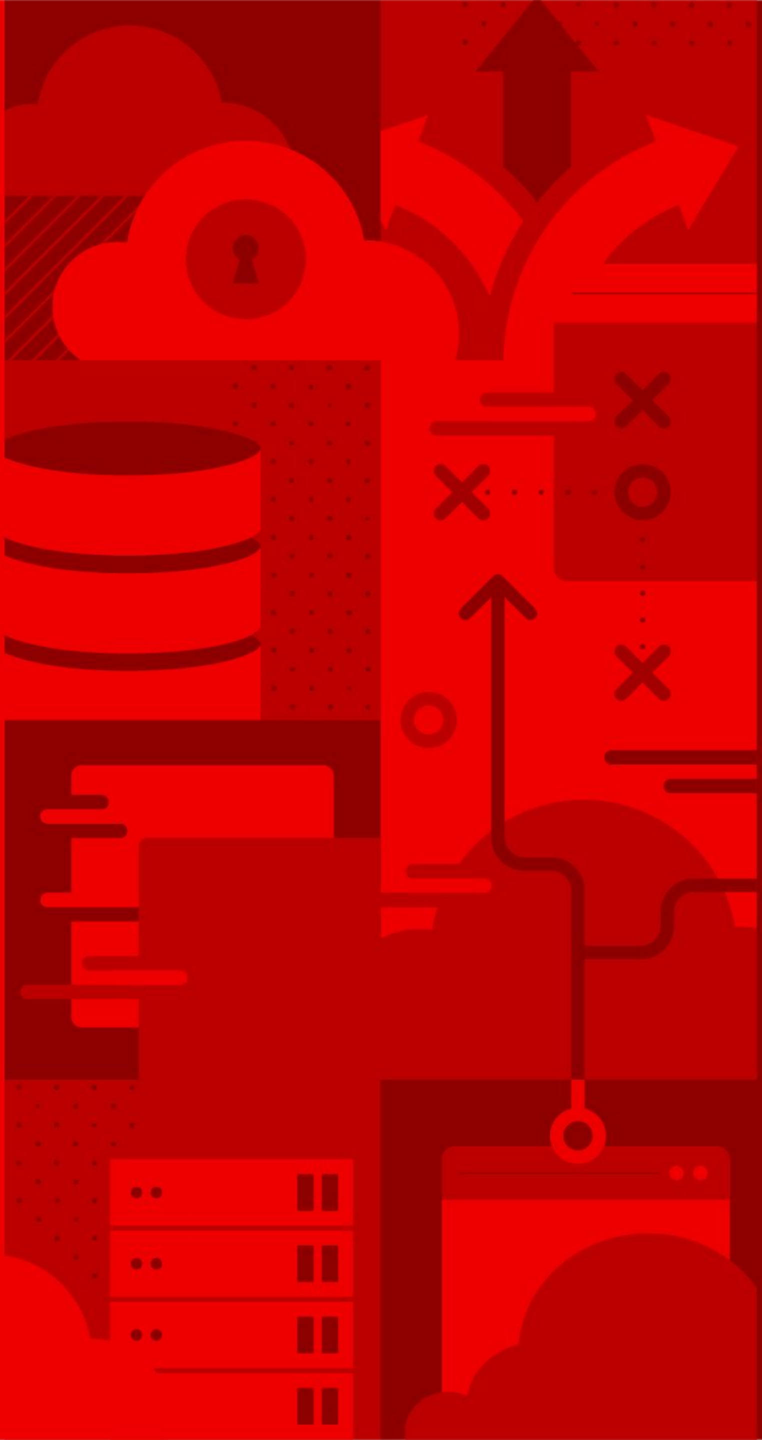
It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

Resource

Controller

Knowledge

Source:
https://coreos.com/blog/introducing-operators.html

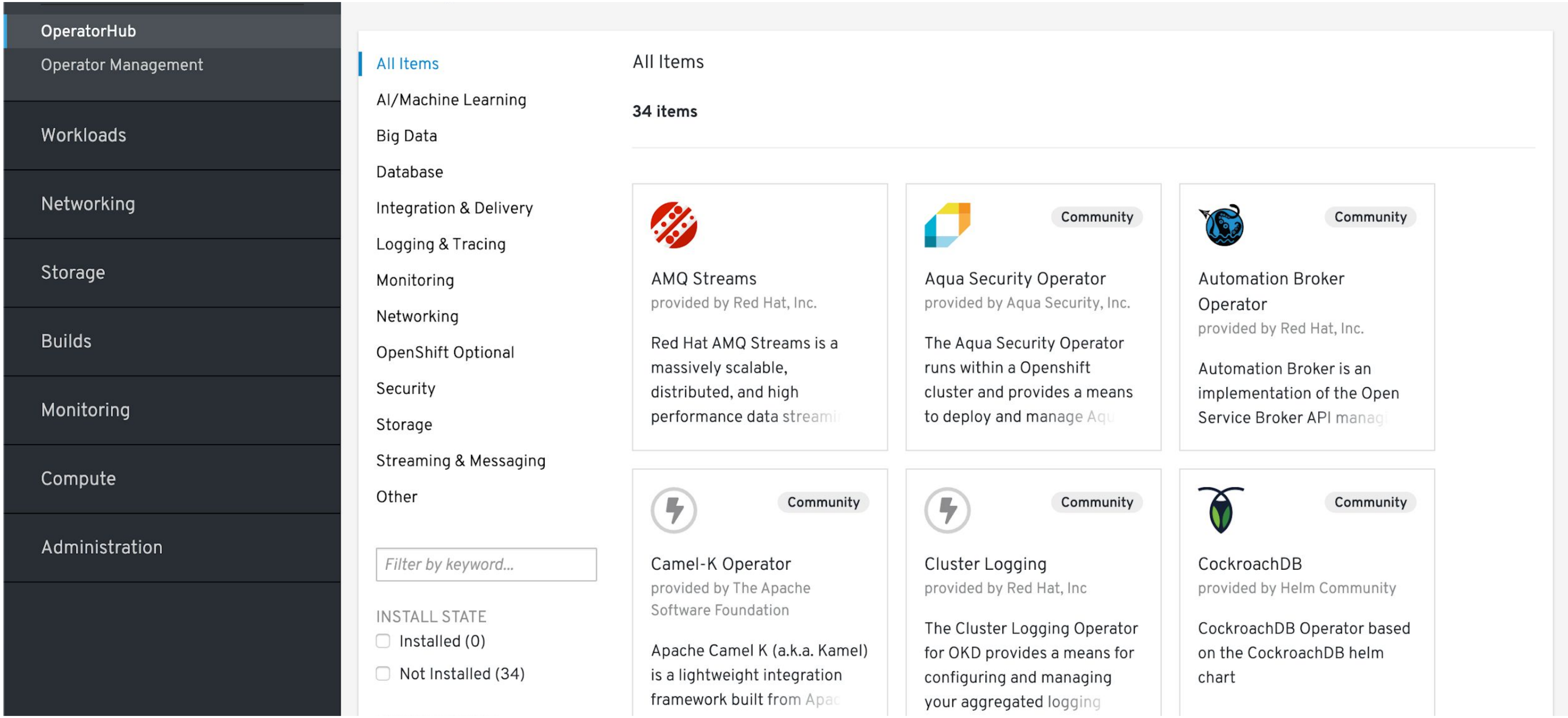


Why do Operators matter to us at Red Hat?

Why Operators Matter to Red Hat

- Build an ecosystem of software on OpenShift that can be as easy, safe, and reliable to use and operate as a Cloud Service.
- Low-touch, remotely managed, one-click-updates.
- Super easy to deploy in an Operator in a Kubernetes environment.

What is an Operator



All Items

34 items

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

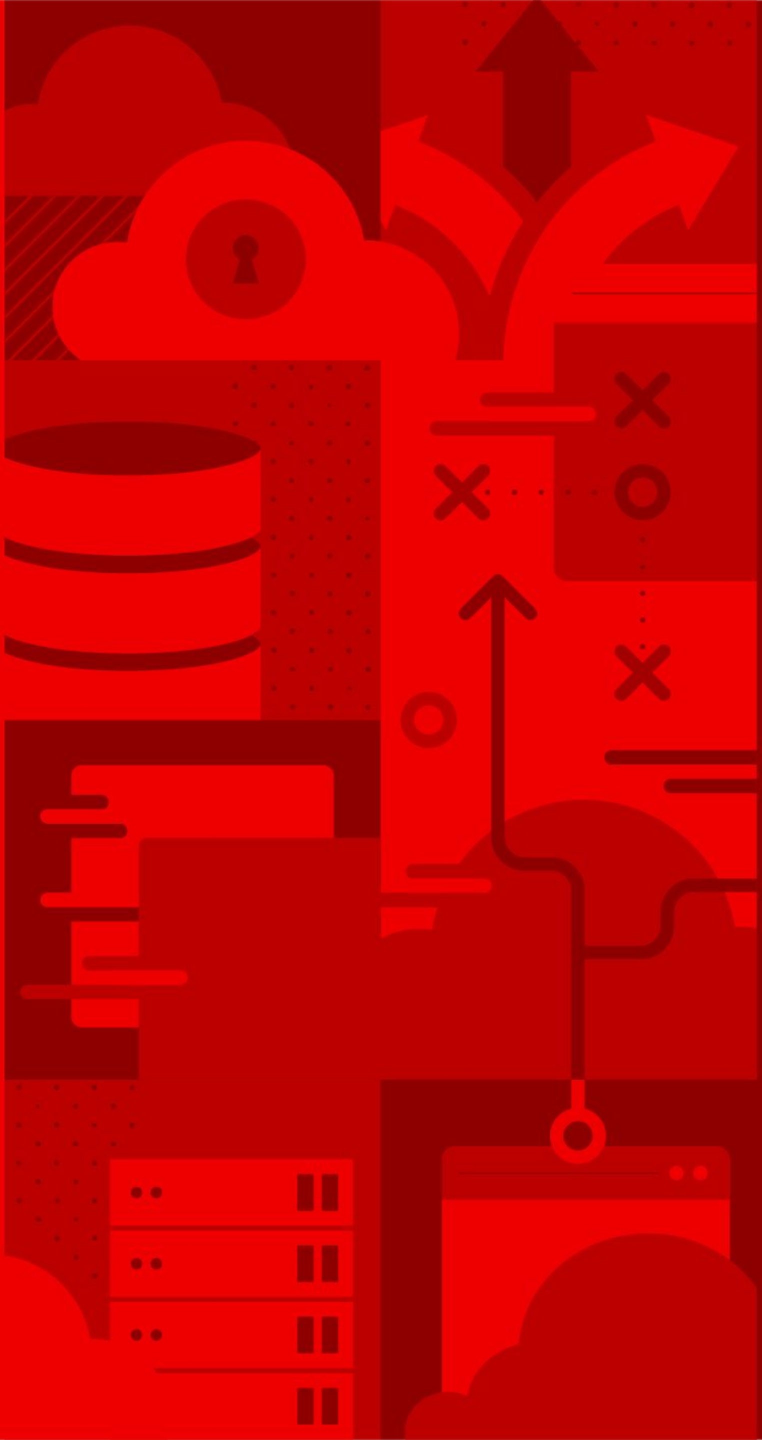
Community
Aqua Security Operator
provided by Aqua Security, Inc.
The Aqua Security Operator runs within a Openshift cluster and provides a means to deploy and manage Aqu

Community
Automation Broker Operator
provided by Red Hat, Inc.
Automation Broker is an implementation of the Open Service Broker API manag

Community
Camel-K Operator
provided by The Apache Software Foundation
Apache Camel K (a.k.a. Kamel) is a lightweight integration framework built from Apac

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

Community
CockroachDB
provided by Helm Community
CockroachDB Operator based on the CockroachDB helm chart



How do I create my very own Operator?

Life Before the Operator SDK

If only it were as simple as...

Resources

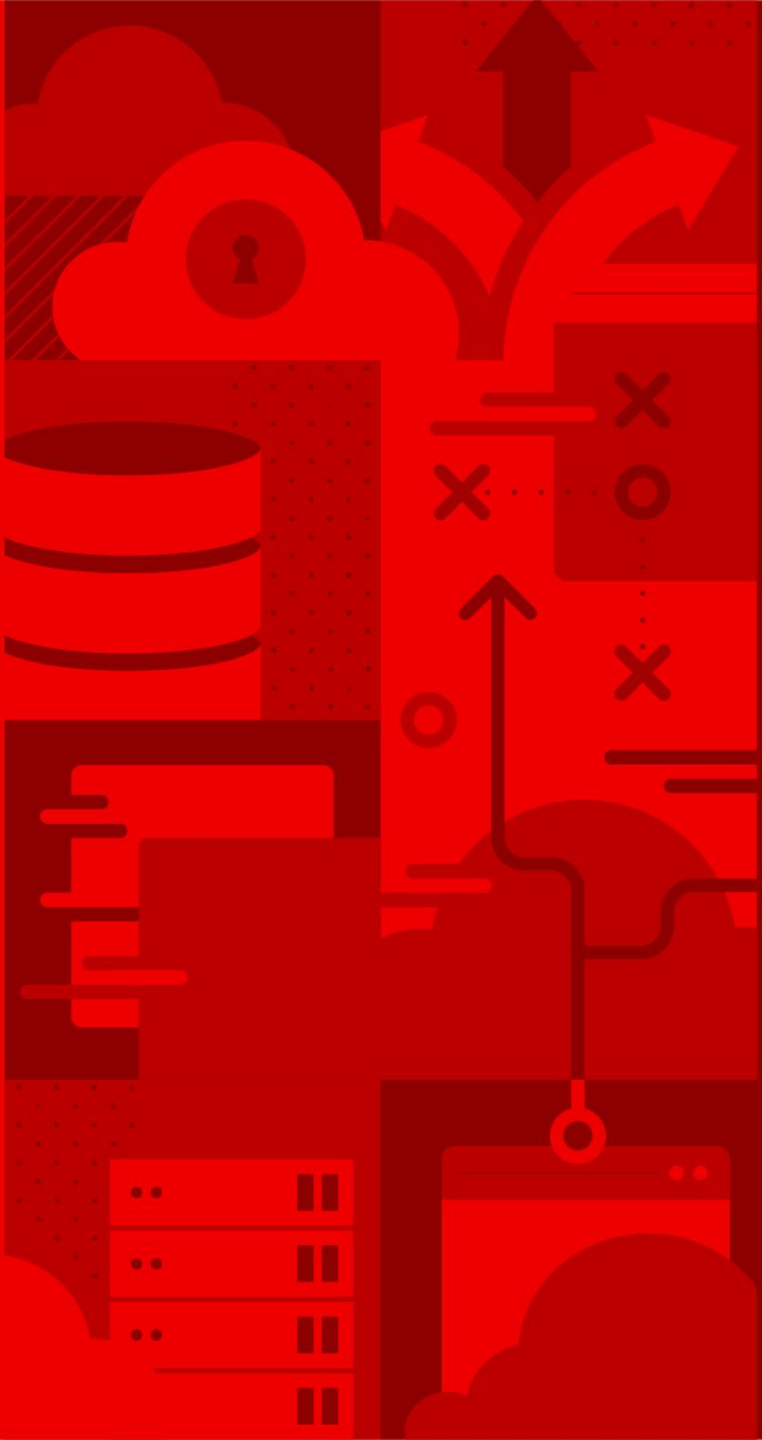
```
type MyCustomResourceDefinition struct {
   // API obj kind & schema version
   metav1.TypeMeta
   // Standard object metadata (optional)
   Metadata api.ObjectMeta
   // Describe how the resource appears
   Spec v1beta1.CustomResourceDefinitionSpec
   // State of the CRD
   Status CustomResourceDefinitionStatus
}
```

Controllers

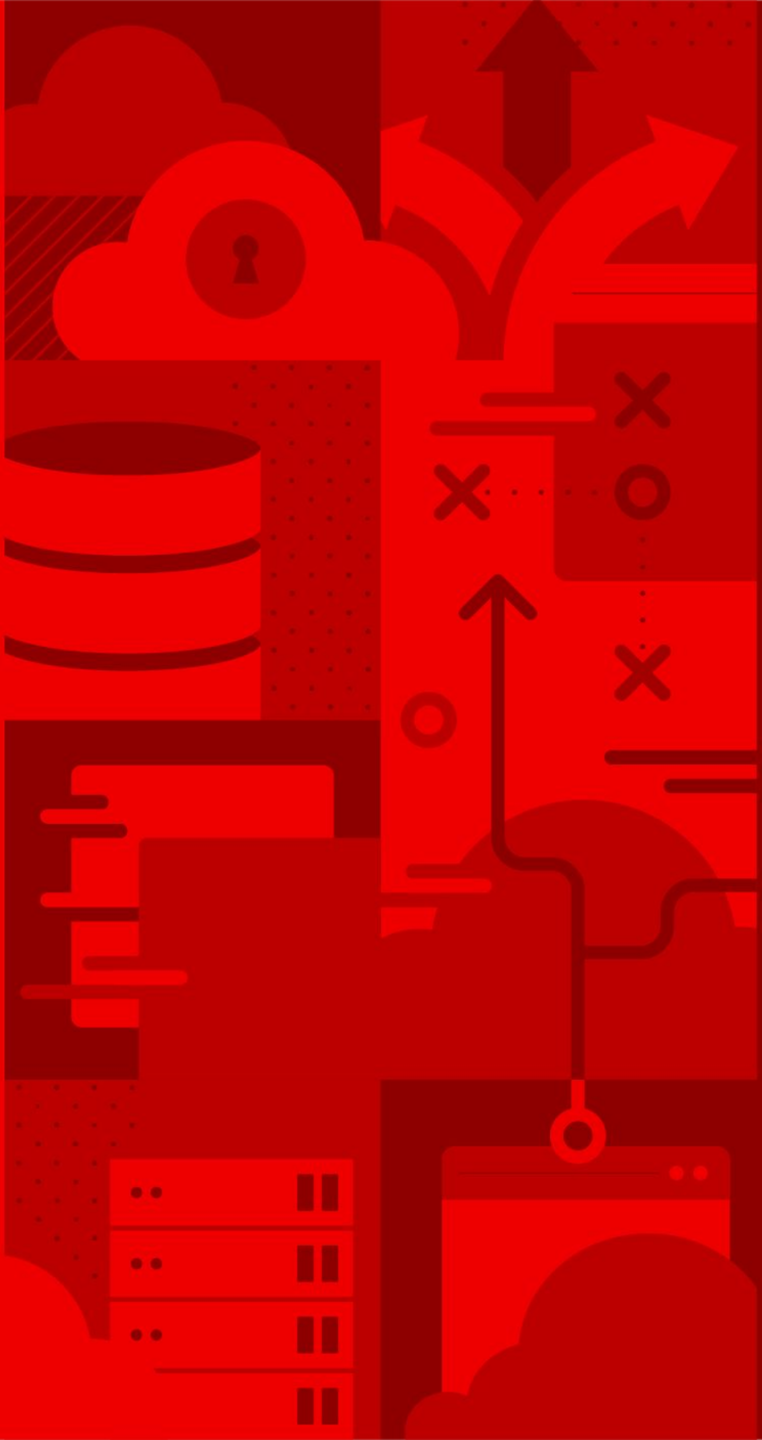
```
for {
   current := getCurrentState()
   desired := getDesiredState()
   makeChanges(current, desired)
}
```

Writing Operator from scratch is Challenging

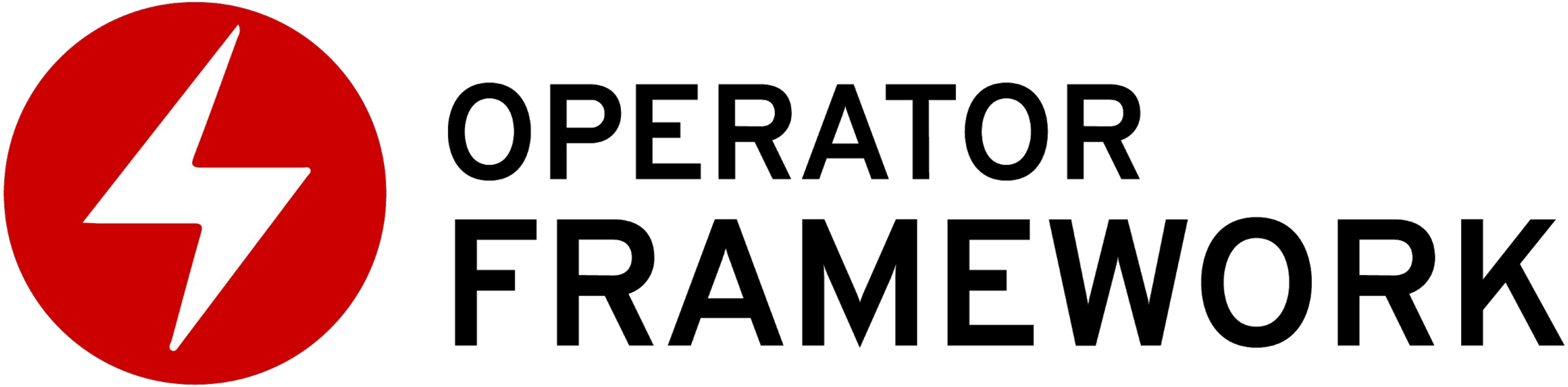
- Research client-library.
- Repo organization.
- Write boiler-plate code.
- Use code generators.
- Knowledge of informers/shared informers and work queues for object cache and event handling.



We need an easier way to **create** Operators

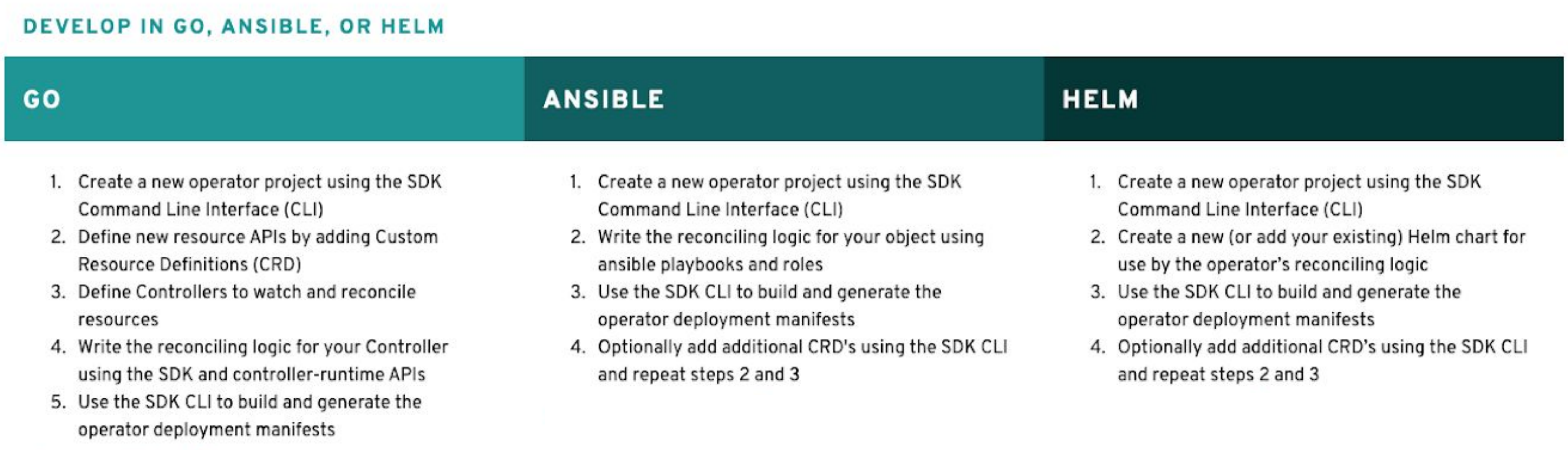


We need an easier way to **manage** Operators



Sources:
https://github.com/operator-framework

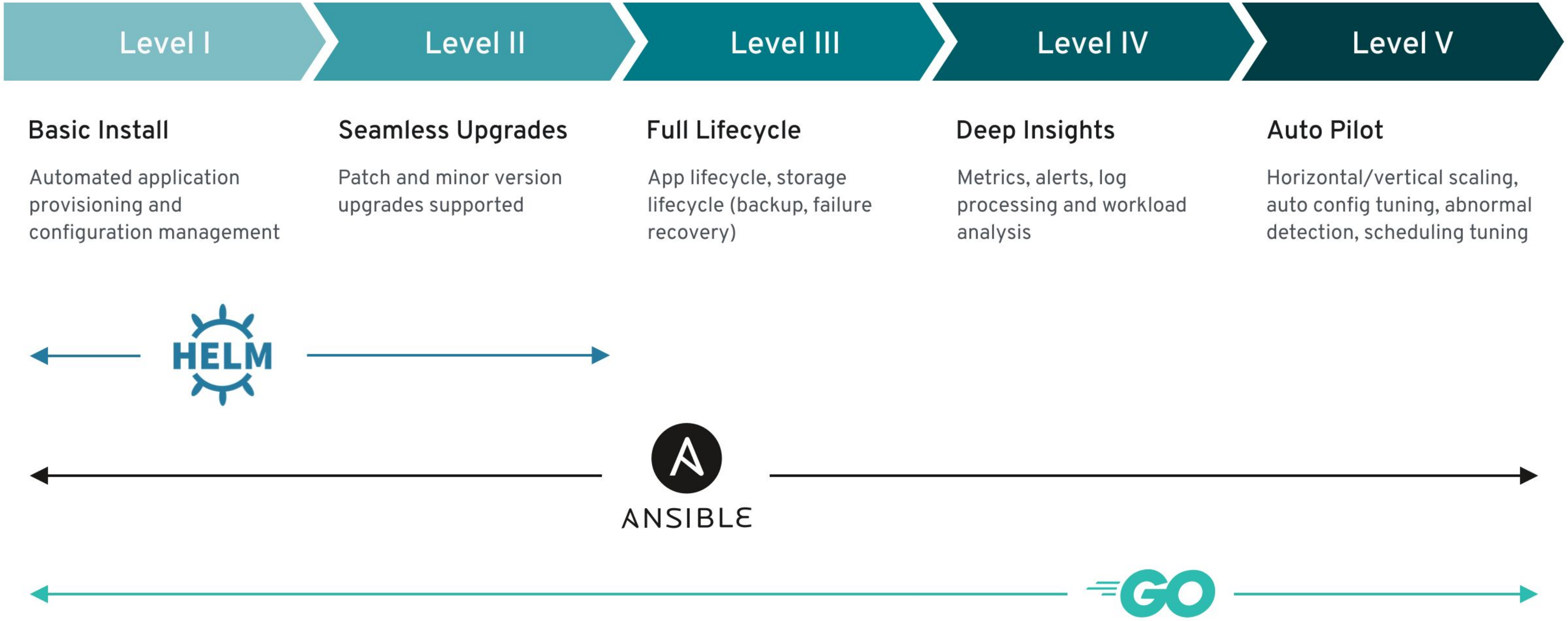
Operator SDK



DEVELOP IN GO, ANSIBLE, OR HELM

GO	ANSIBLE	HELM
1. Create a new operator project using the SDK Command Line Interface (CLI) 2. Define new resource APIs by adding Custom Resource Definitions (CRD) 3. Define Controllers to watch and reconcile resources 4. Write the reconciling logic for your Controller using the SDK and controller-runtime APIs 5. Use the SDK CLI to build and generate the operator deployment manifests	1. Create a new operator project using the SDK Command Line Interface (CLI) 2. Write the reconciling logic for your object using ansible playbooks and roles 3. Use the SDK CLI to build and generate the operator deployment manifests 4. Optionally add additional CRD's using the SDK CLI and repeat steps 2 and 3	1. Create a new operator project using the SDK Command Line Interface (CLI) 2. Create a new (or add your existing) Helm chart for use by the operator's reconciling logic 3. Use the SDK CLI to build and generate the operator deployment manifests 4. Optionally add additional CRD's using the SDK CLI and repeat steps 2 and 3

Operator SDK



Level I

Basic Install

Automated application provisioning and configuration management

Level II

Seamless Upgrades

Patch and minor version upgrades supported

Level III

Full Lifecycle

App lifecycle, storage lifecycle (backup, failure recovery)

Level IV

Deep Insights

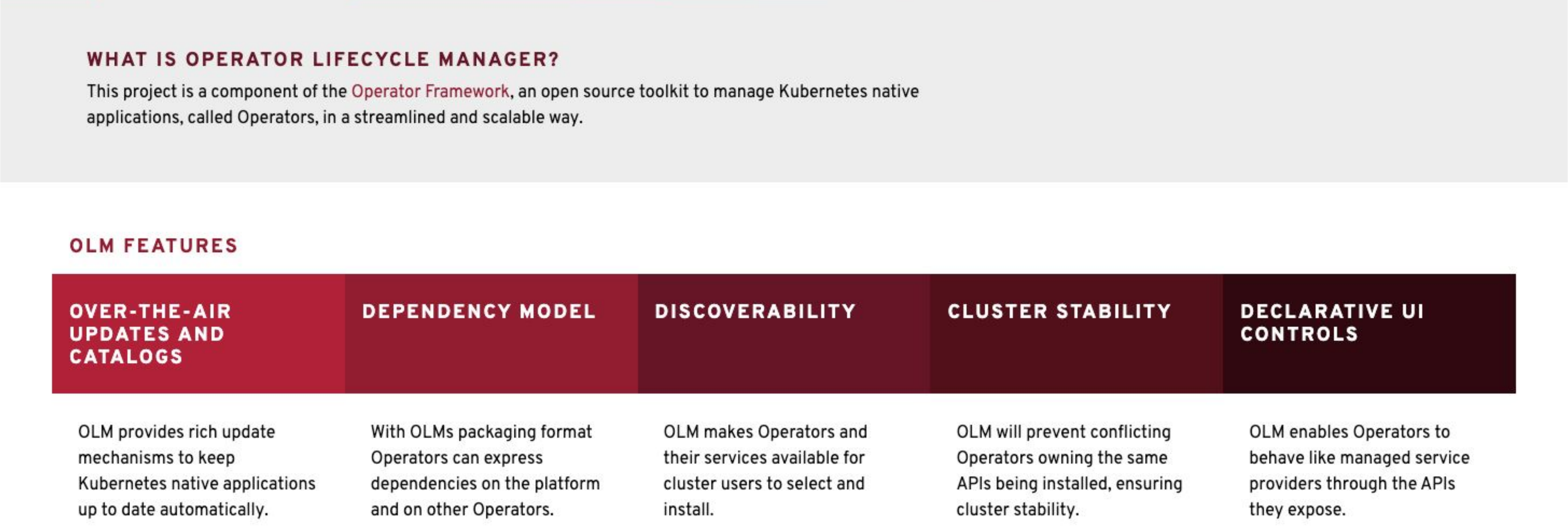
Metrics, alerts, log processing and workload analysis

Level V

Auto Pilot

Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

Operator Lifecycle Manager



WHAT IS OPERATOR LIFECYCLE MANAGER?

This project is a component of the Operator Framework, an open source toolkit to manage Kubernetes native applications, called Operators, in a streamlined and scalable way.

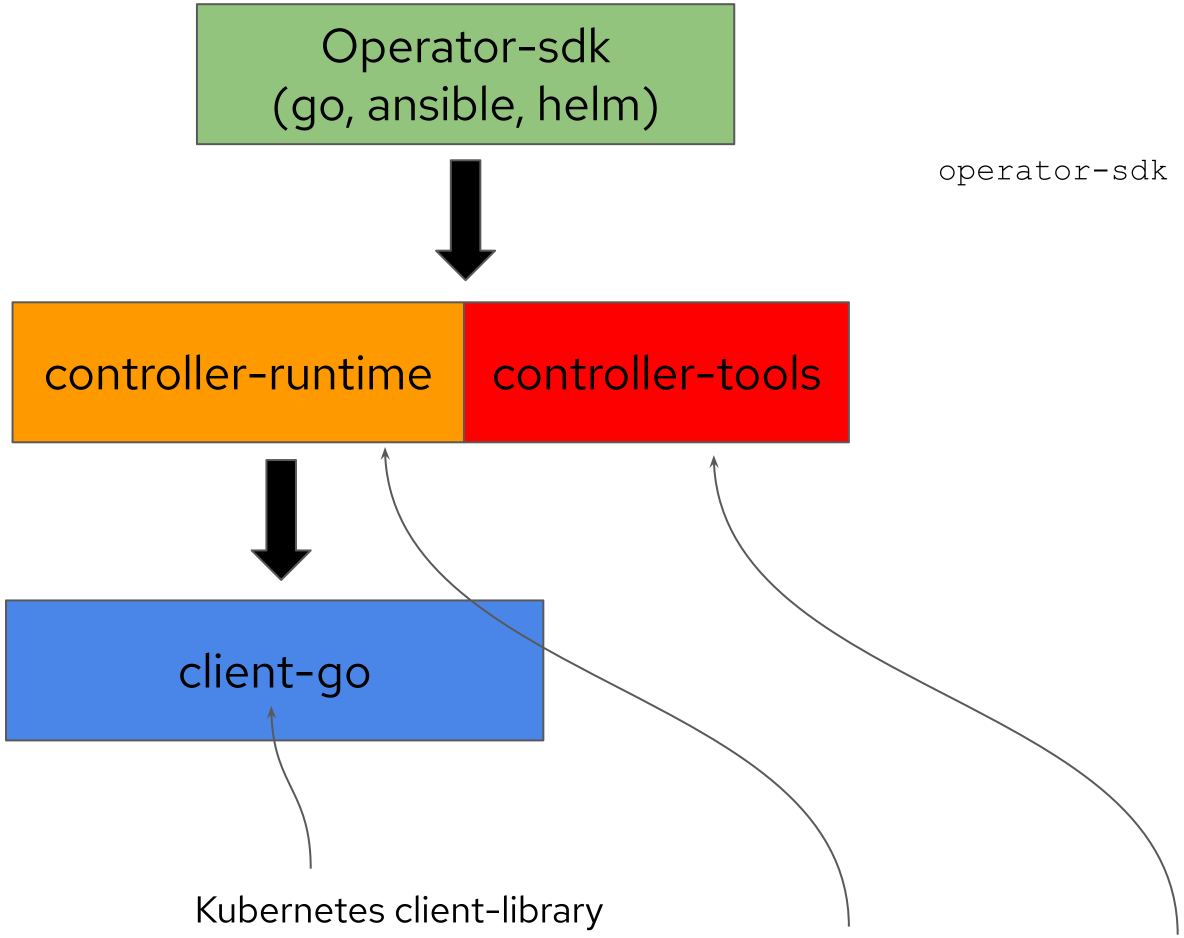
OLM FEATURES

OVER-THE-AIR UPDATES AND CATALOGS	DEPENDENCY MODEL	DISCOVERABILITY	CLUSTER STABILITY	DECLARATIVE UI CONTROLS
OLM provides rich update mechanisms to keep Kubernetes native applications up to date automatically.	With OLMs packaging format Operators can express dependencies on the platform and on other Operators.	OLM makes Operators and their services available for cluster users to select and install.	OLM will prevent conflicting Operators owning the same APIs being installed, ensuring cluster stability.	OLM enables Operators to behave like managed service providers through the APIs they expose.

About Operator-SDK

How things were before..

Operator-SDK (released in 2018 by RedHat)



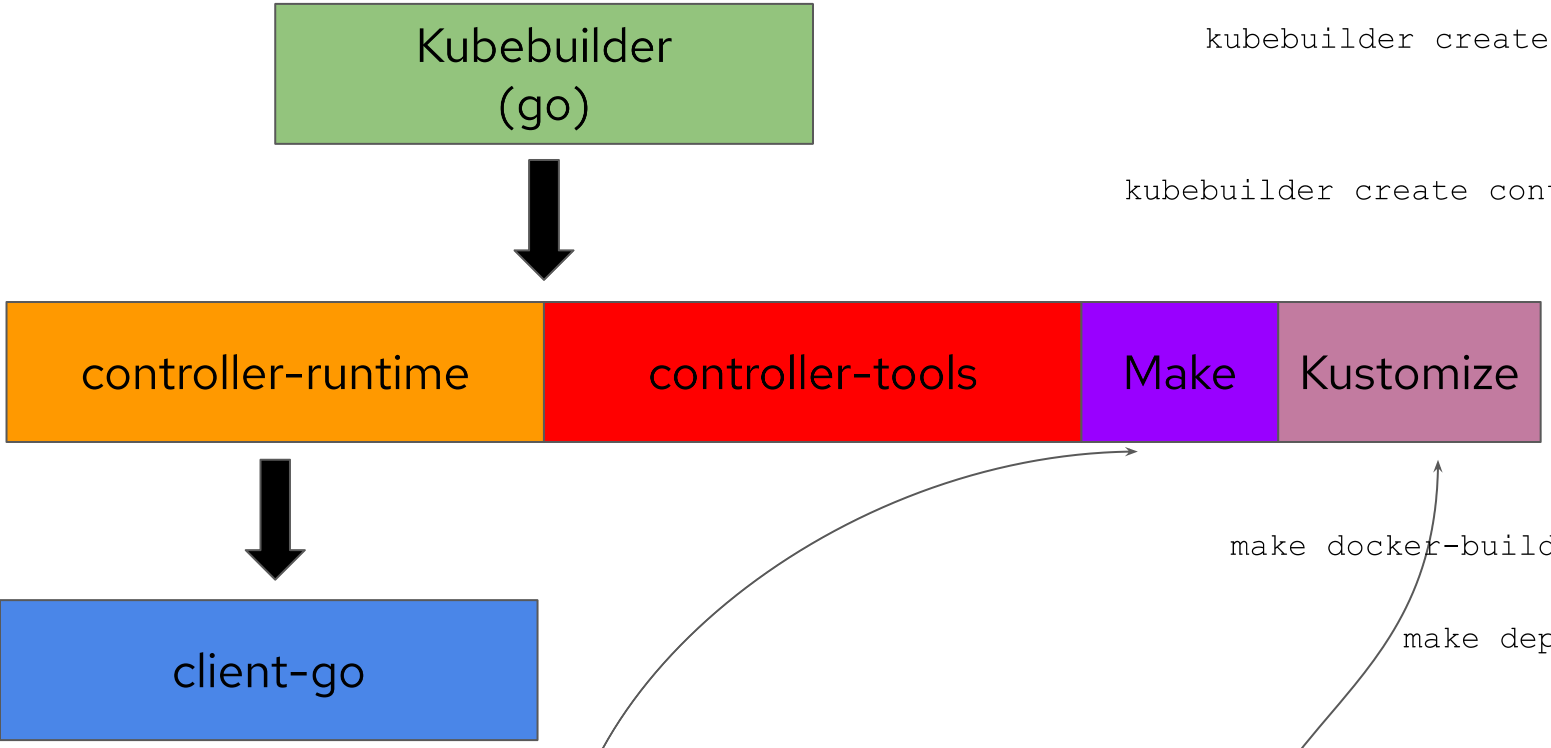
Kubernetes client-library

Libraries for building the controller part of your operator

Tools for generating custom resource definitions, rbac artifacts, and more!

```
operator-sdk new create app-operator --type=go

operator-sdk add api --api-version=app.example.com/v1alpha1 --kind=App

operator-sdk generate k8s

operator-sdk generate crds

operator-sdk add controller --api-version=app.example.com/v1alpha1 --kind=App

operator-sdk run --local --kubeconfig=

operator-sdk build quay.io/example/operator:v0.0.1

podman push quay.io/example/operator:v0.0.1

operator-sdk olm install

operator-sdk bundle create quay.io/example/operator:v0.0.1 \
    --directory ./deploy/olm-catalog/test-operator \
    --package test-operator \
    --channels stable,beta \
    --default-channel stable

podman build -t quay.io/example/operator-bundle:v0.0.1 -f
upstream-example.Dockerfile .

podman push quay.io/example/operator-bundle:v0.0.1
```

Kubebuilder (released in 2018 by API Machinery group)



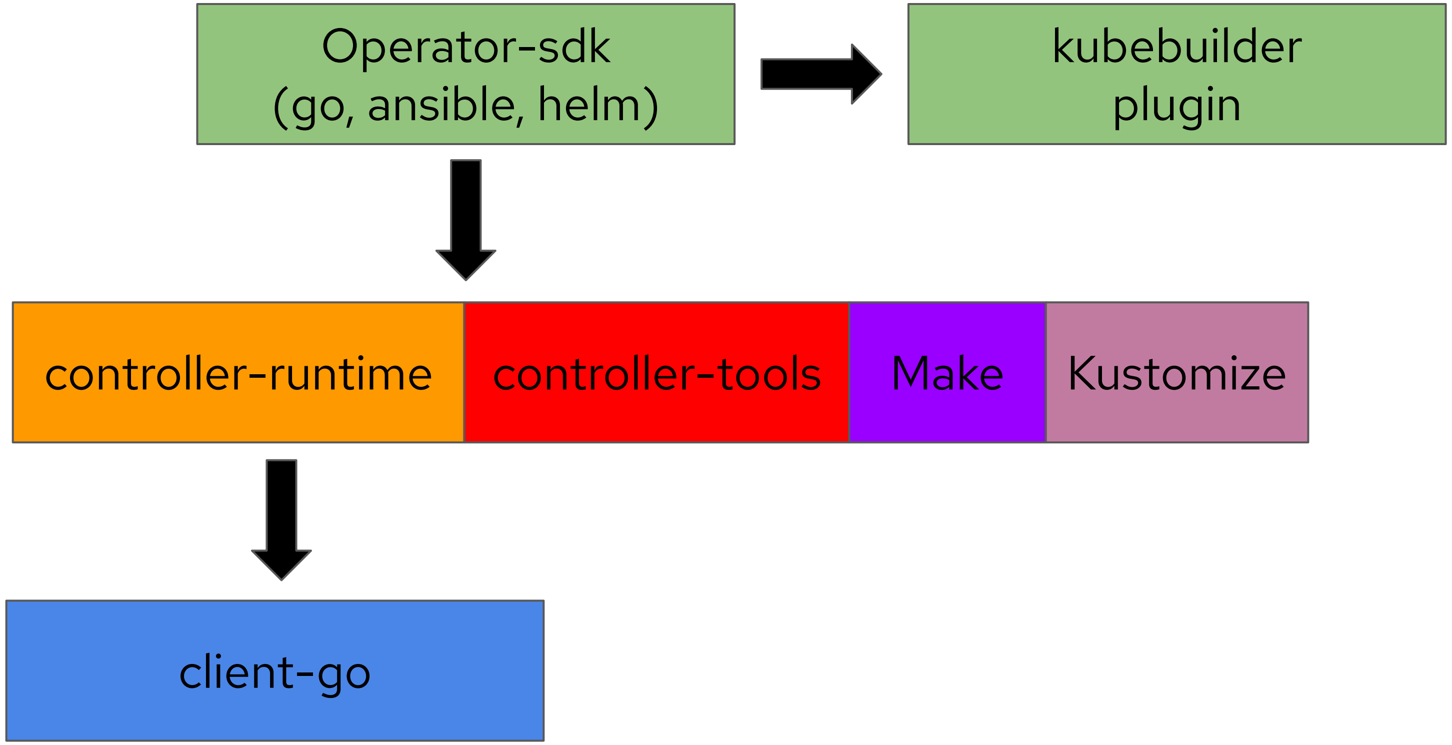
```
kubebuilder init --domain my.domain

kubebuilder create api --group webapp --version v1 --kind Guestbook

make manifests

kubebuilder create controller --group webapp --version v1 --kind Guestbook

make install

make run

kubectl apply -f config/samples/

make docker-build docker-push IMG=<some-registry>/<project-name>:tag

make deploy IMG=<some-registry>/<project-name>:tag
```

Provides commands to test, run, build, generate etc.
Easy to customize!

Allows you to customize your kube Yaml
objects without templating.

Now with Operator-SDK 1.0.0...

Operator-SDK (1.0.0)



```
operator-sdk init --domain my.domain
```

```
operator-sdk create api --group webapp \
--version v1 --kind Guestbook
```

```
make manifests
```

```
operator-sdk create controller --group webapp \
--version v1 --kind Guestbook
```

```
make install
```

etc...

What else is new?

Separate binaries for Go, Ansible, and Helm

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

Support for Webhooks

```
operator-sdk create webhook --group batch --version v1 --kind CronJob --defaulting --programmatic-validation
```

Use Kustomize!

```
kustomize build config/manifests | operator-sdk generate bundle --overwrite --version 0.0.1
```

Other stuff...

Live Demo…but first..some review..

Resource Schema Components

4

GVK aka TypeMeta

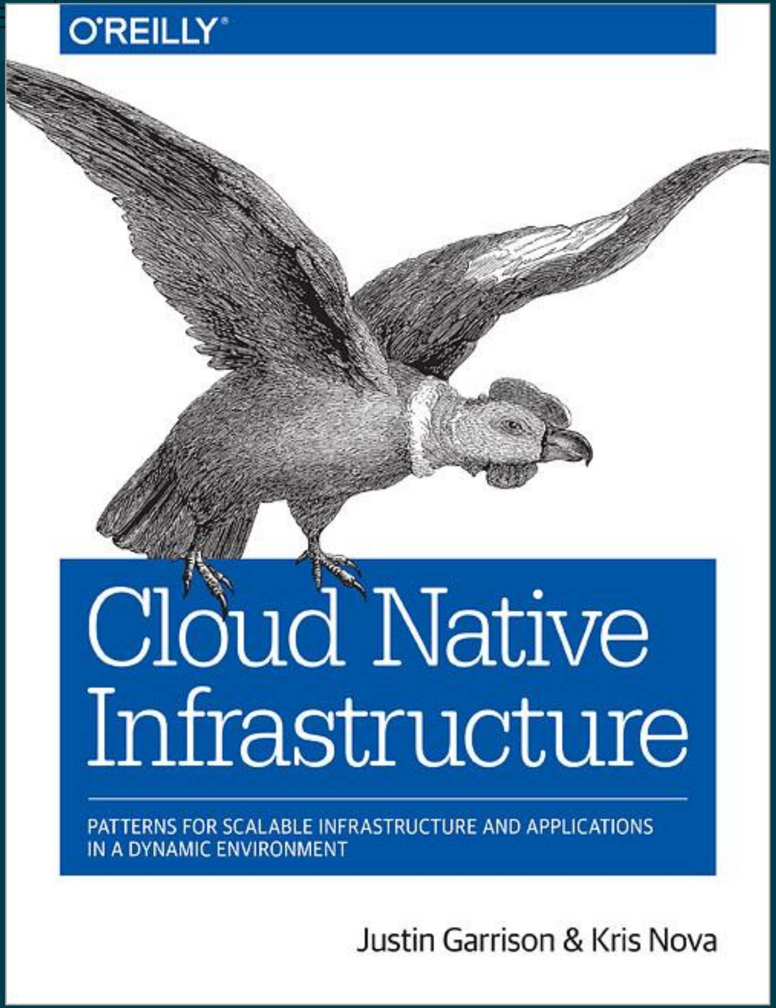
Metadata aka ObjectMeta

Spec

Status

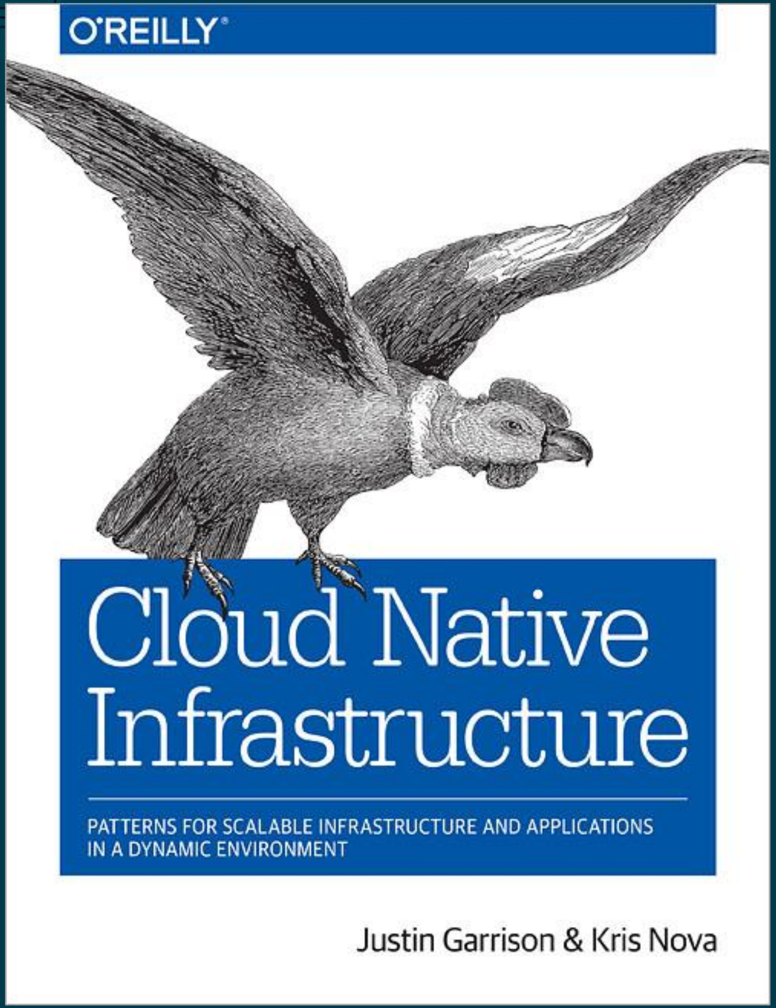
```yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet

metadata:
  name: my-first-replica-set
  namespace: myproject

spec:
  selector:
    matchLabels:
     app: nginx
  replicas: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
status:
  availableReplicas: 1
  fullyLabeledReplicas: 1
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
```

Red Hat


O'REILLY
Cloud Native Infrastructure
PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT
Justin Garrison & Kris Nova

Chapter 4
Designing Infrastructure Applications

The ***reconciler pattern*** *is a software pattern that can be used or expanded upon for managing cloud native infrastructure. The pattern enforces the idea of having two representations of the infrastructure—the first being the actual state of the infrastructure, and the second being the expected state of the infrastructure.*



O'REILLY®

Cloud Native Infrastructure

PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT

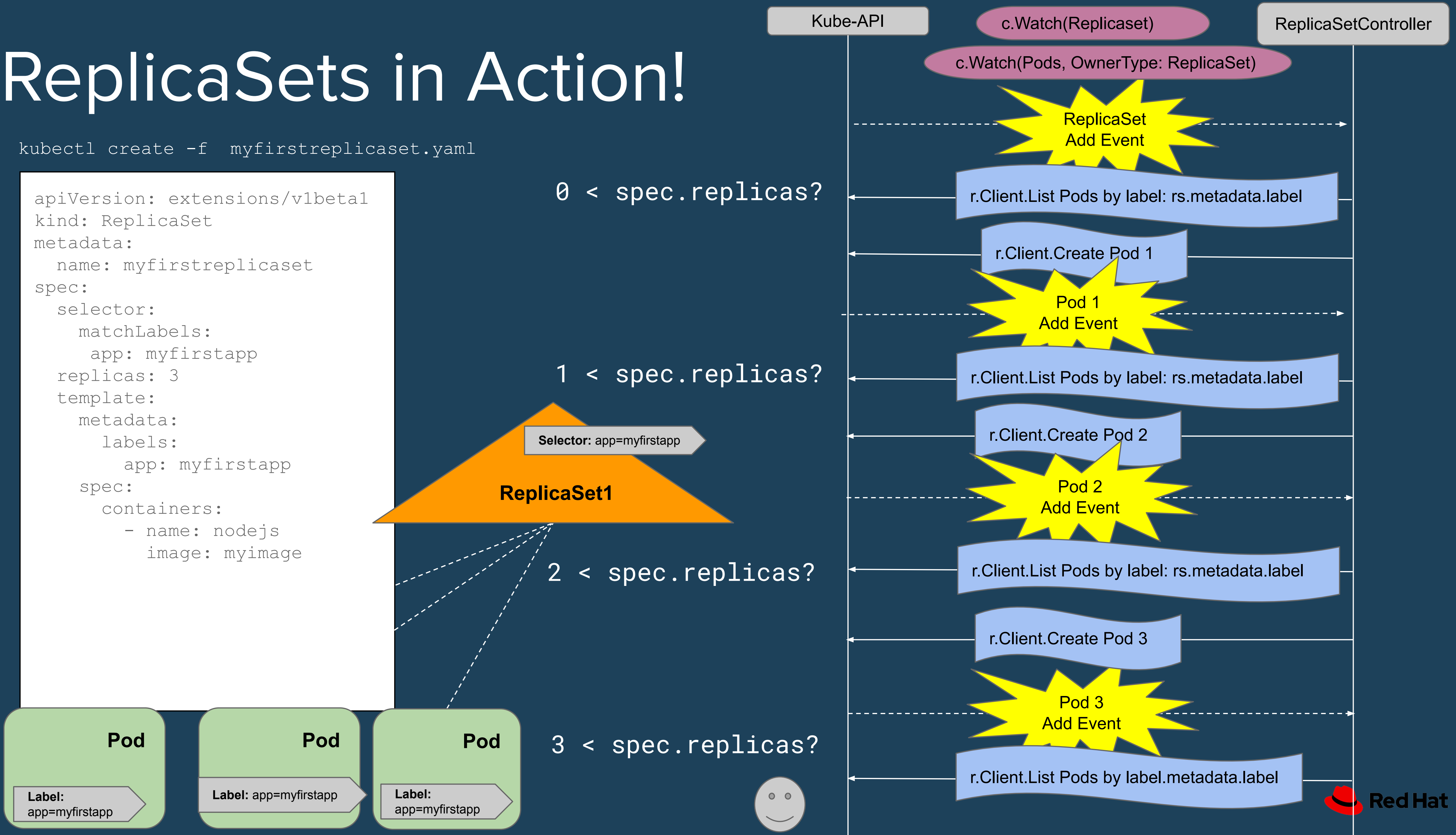
Justin Garrison & Kris Nova

*The **reconciler pattern** will force the engineer to have two independent avenues for getting either of these representations, as well as to implement a solution to reconcile the actual state into the expected state.*

ReplicaSets in Action!

```
kubectl create -f  myfirstreplicaset.yaml
```

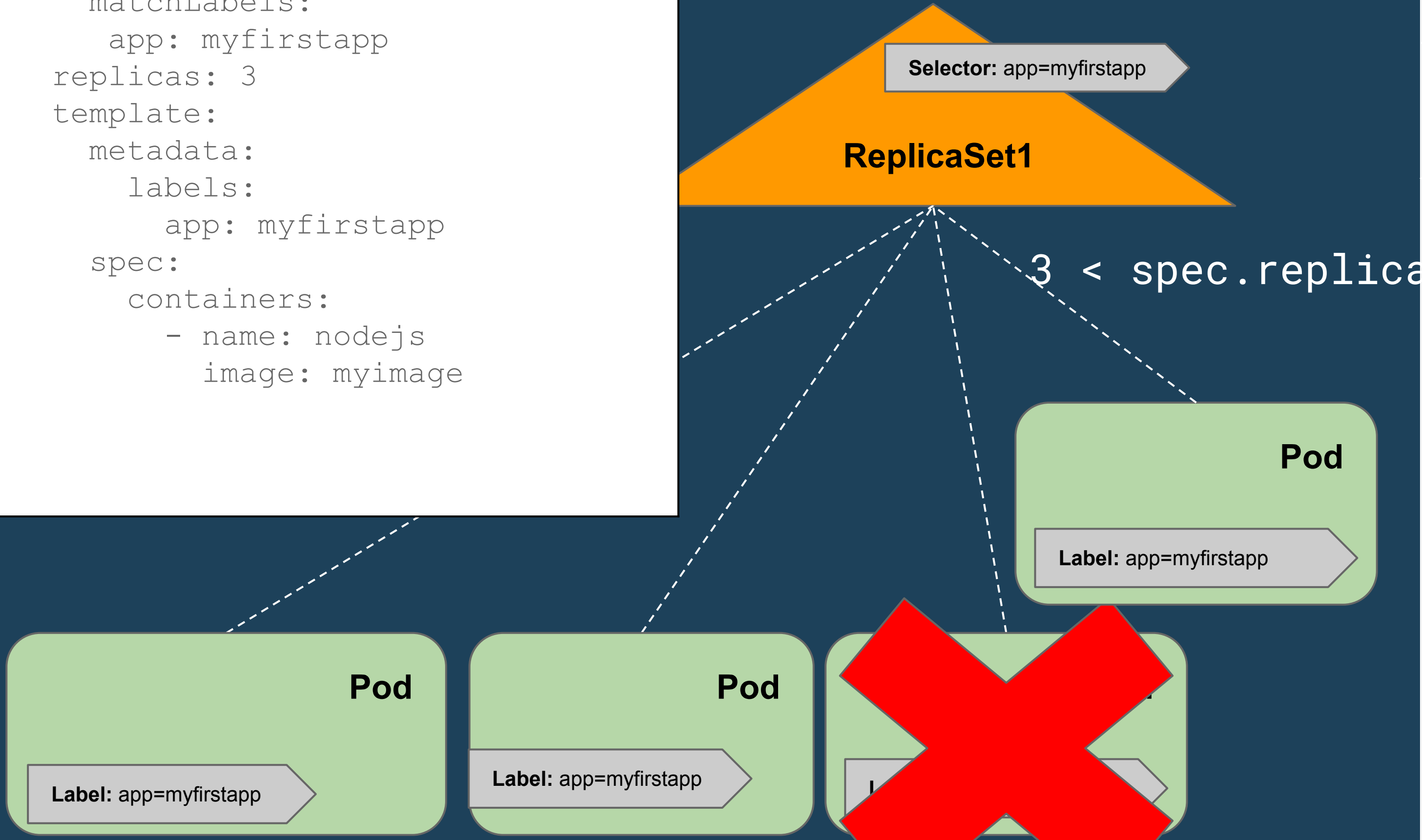
```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```



ReplicaSets in Action!

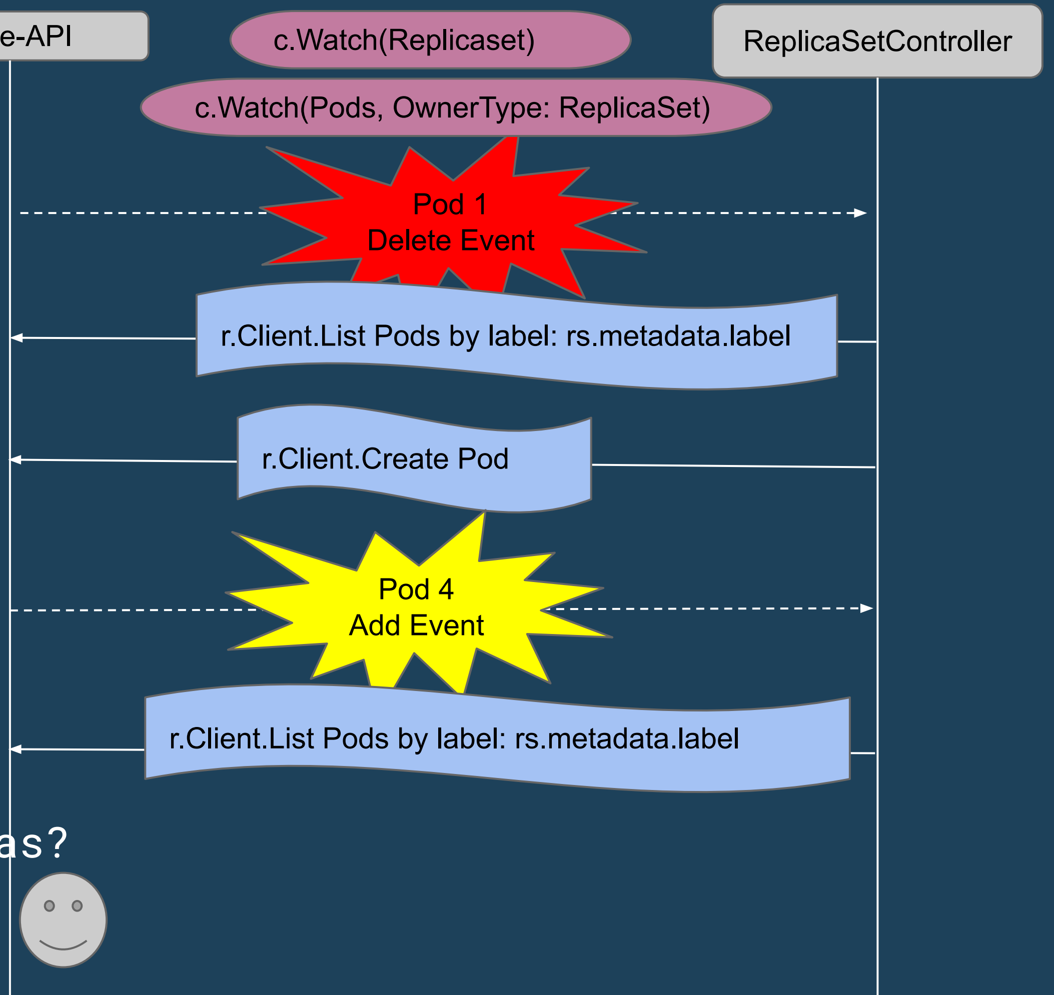
```
kubectl create -f  myfirstreplicaset.yaml
```

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: myfirstreplicaset
spec:
  selector:
    matchLabels:
     app: myfirstapp
  replicas: 3
  template:
    metadata:
      labels:
        app: myfirstapp
    spec:
      containers:
        - name: nodejs
          image: myimage
```



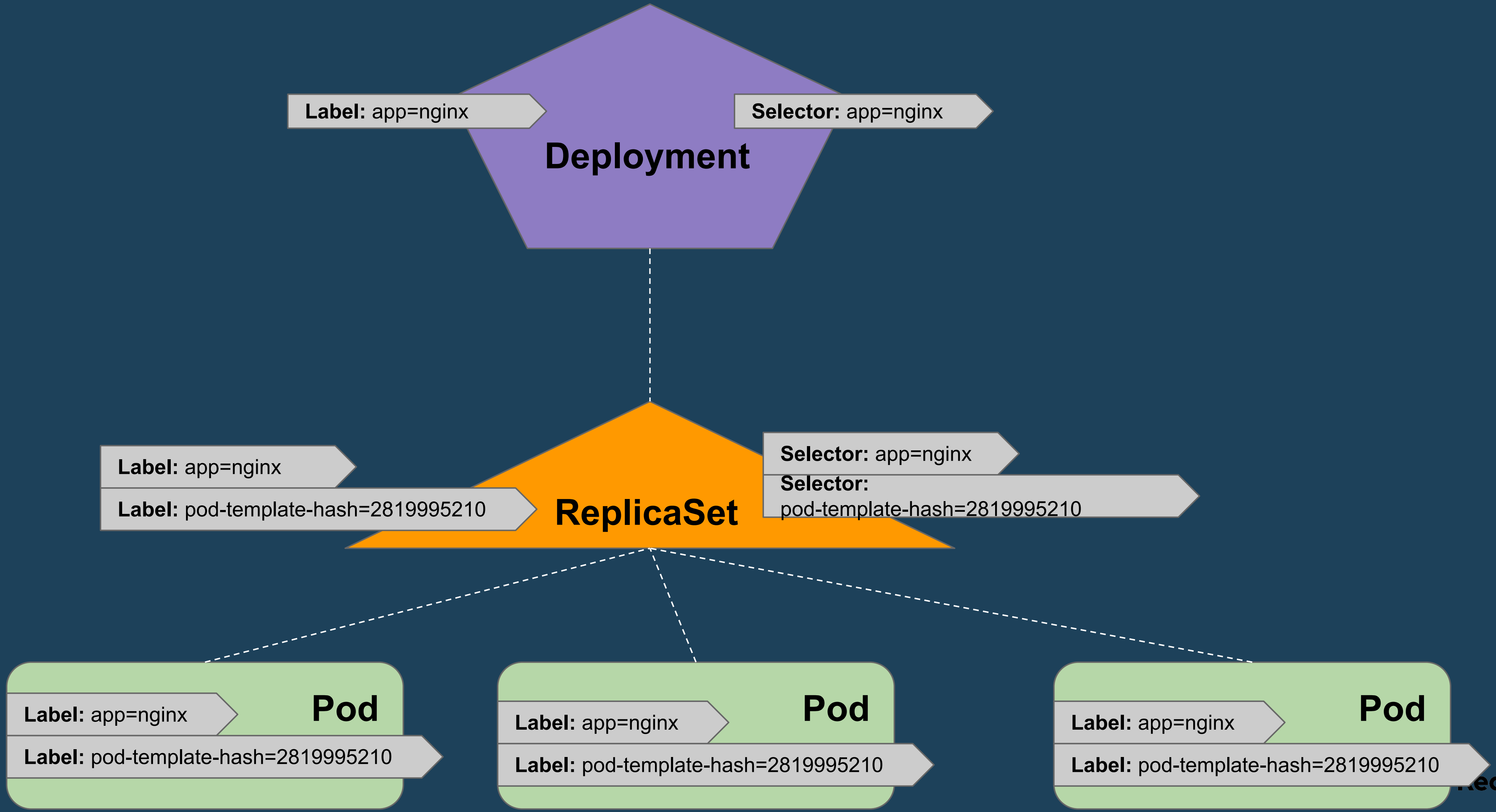
2 < spec.replicas?

3 < spec.replicas?



Kube-API
c.Watch(Replicaset)
ReplicaSetController
c.Watch(Pods, OwnerType: ReplicaSet)
Pod 1
Delete Event
r.Client.List Pods by label: rs.metadata.label
r.Client.Create Pod
Pod 4
Add Event
r.Client.List Pods by label: rs.metadata.label

Deployments!



Deployment

Label: app=nginx

Selector: app=nginx

ReplicaSet

Label: app=nginx

Label: pod-template-hash=2819995210

Selector: app=nginx

Selector: pod-template-hash=2819995210

Pod

Label: app=nginx

Label: pod-template-hash=2819995210

Pod

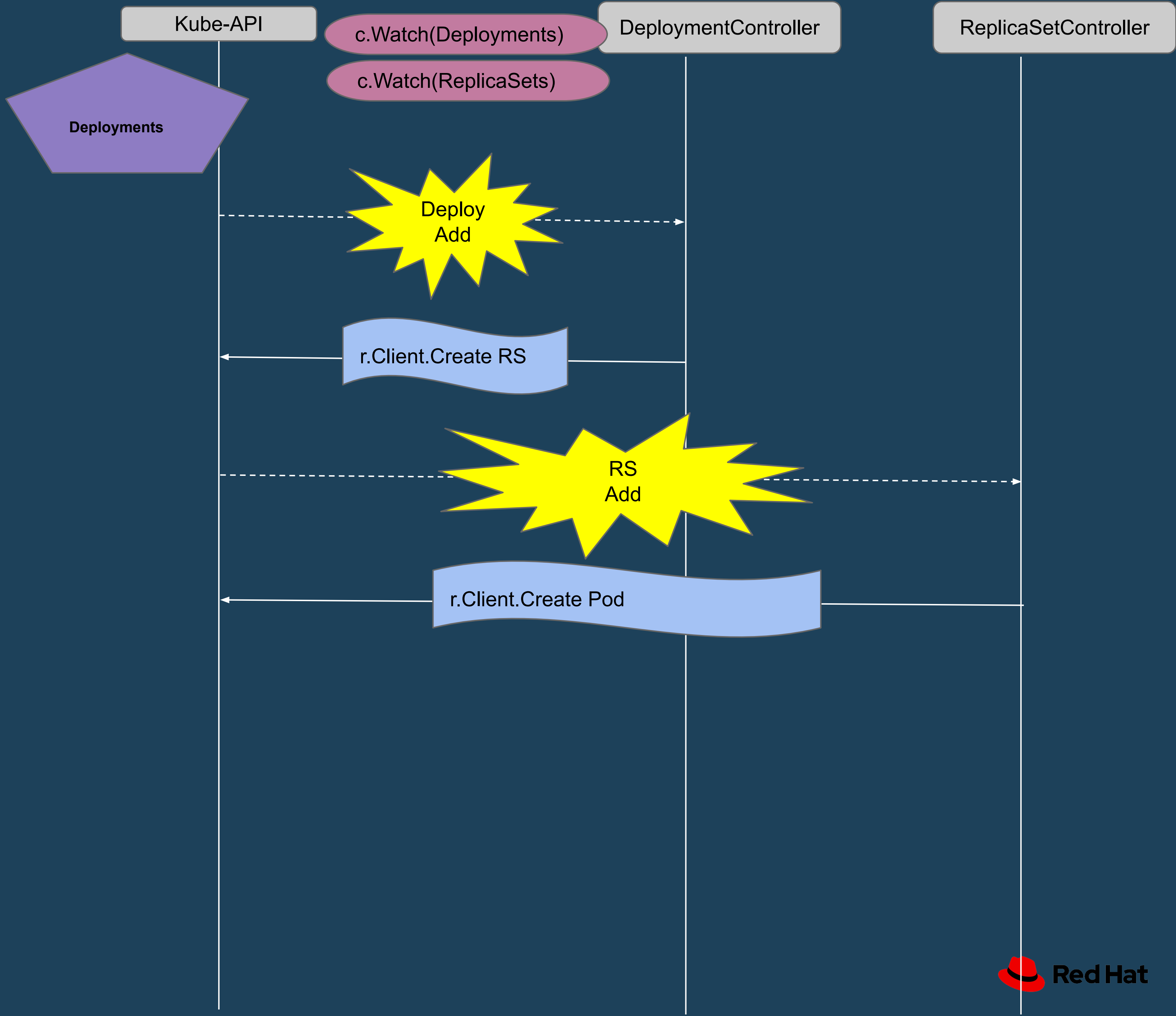
Label: app=nginx

Label: pod-template-hash=2819995210

Pod

Label: app=nginx

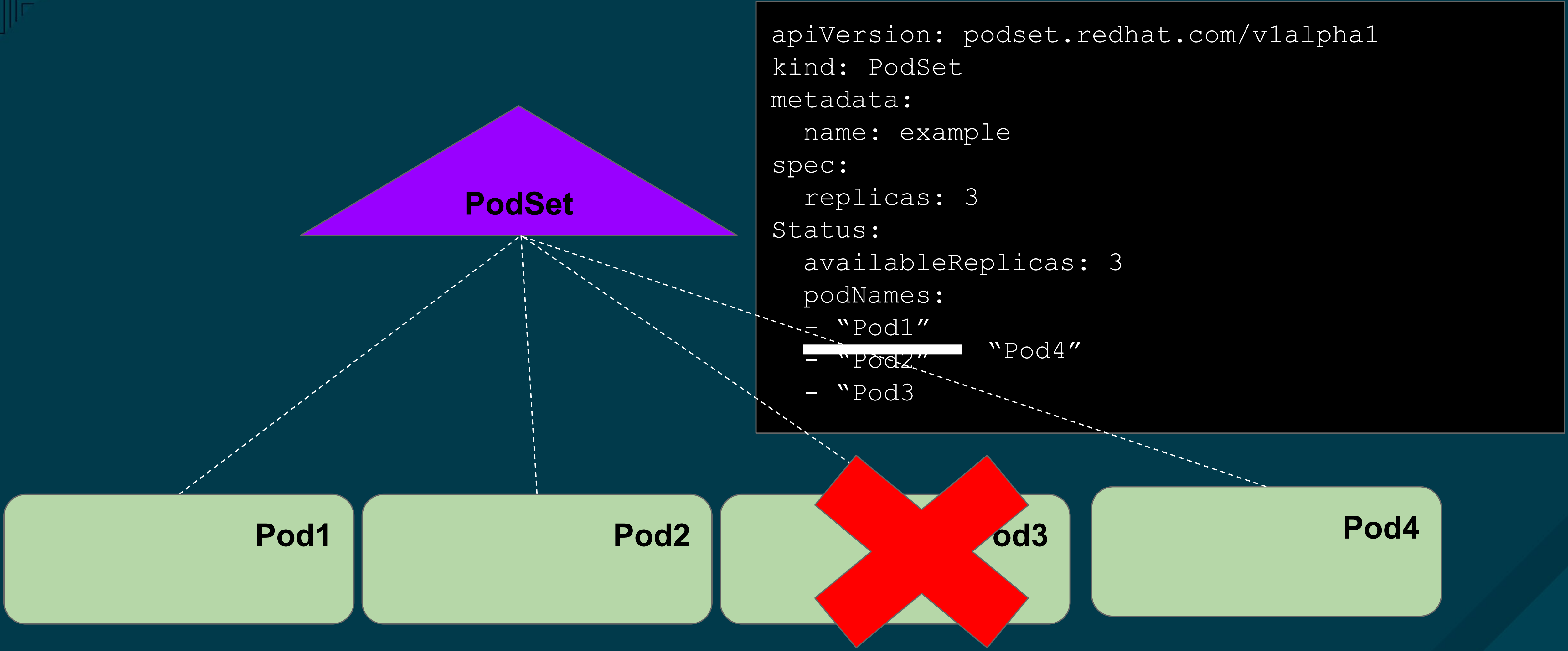
Label: pod-template-hash=2819995210


Kube-API
c.Watch(Deployments)
c.Watch(ReplicaSets)
DeploymentController
ReplicaSetController
Deployments
Deploy Add
r.Client.Create RS
RS Add
r.Client.Create Pod

Red Hat

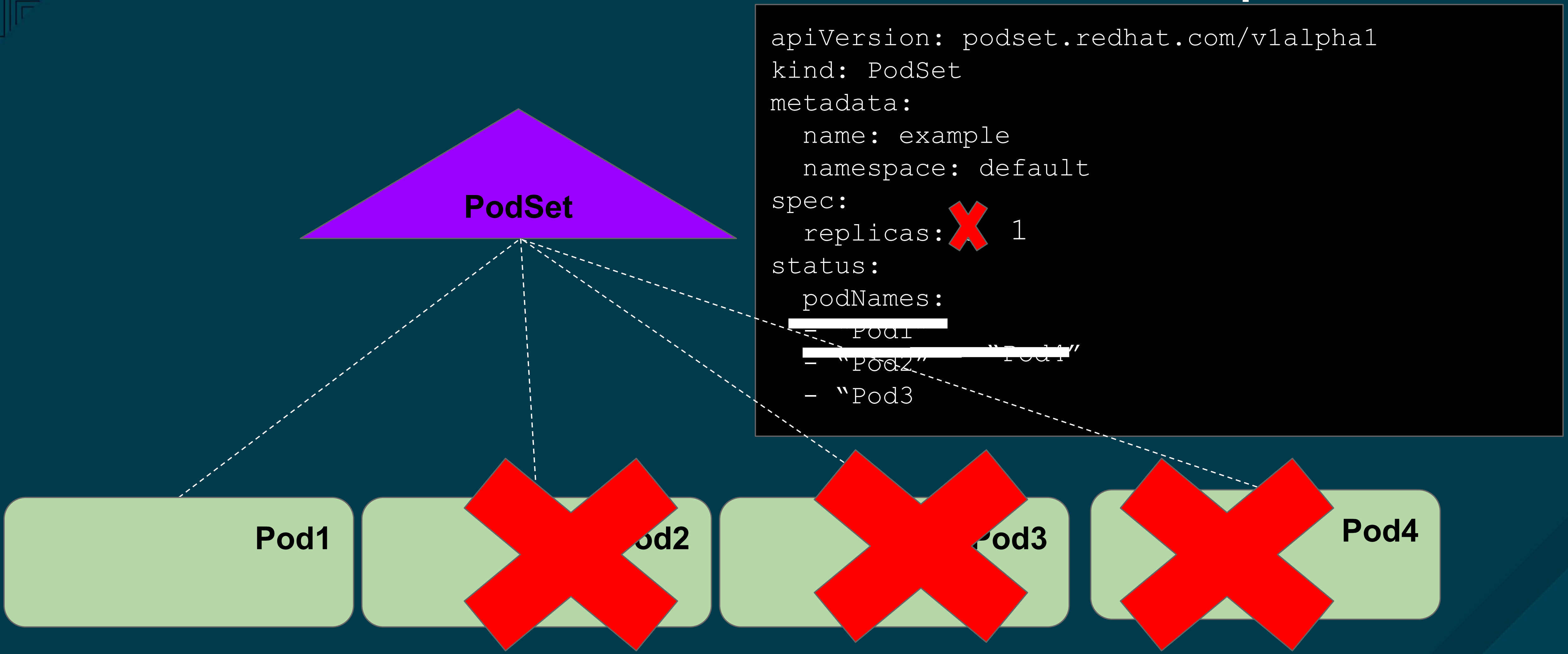
PodSet Operator

A Simple Controller that Manages Pods.

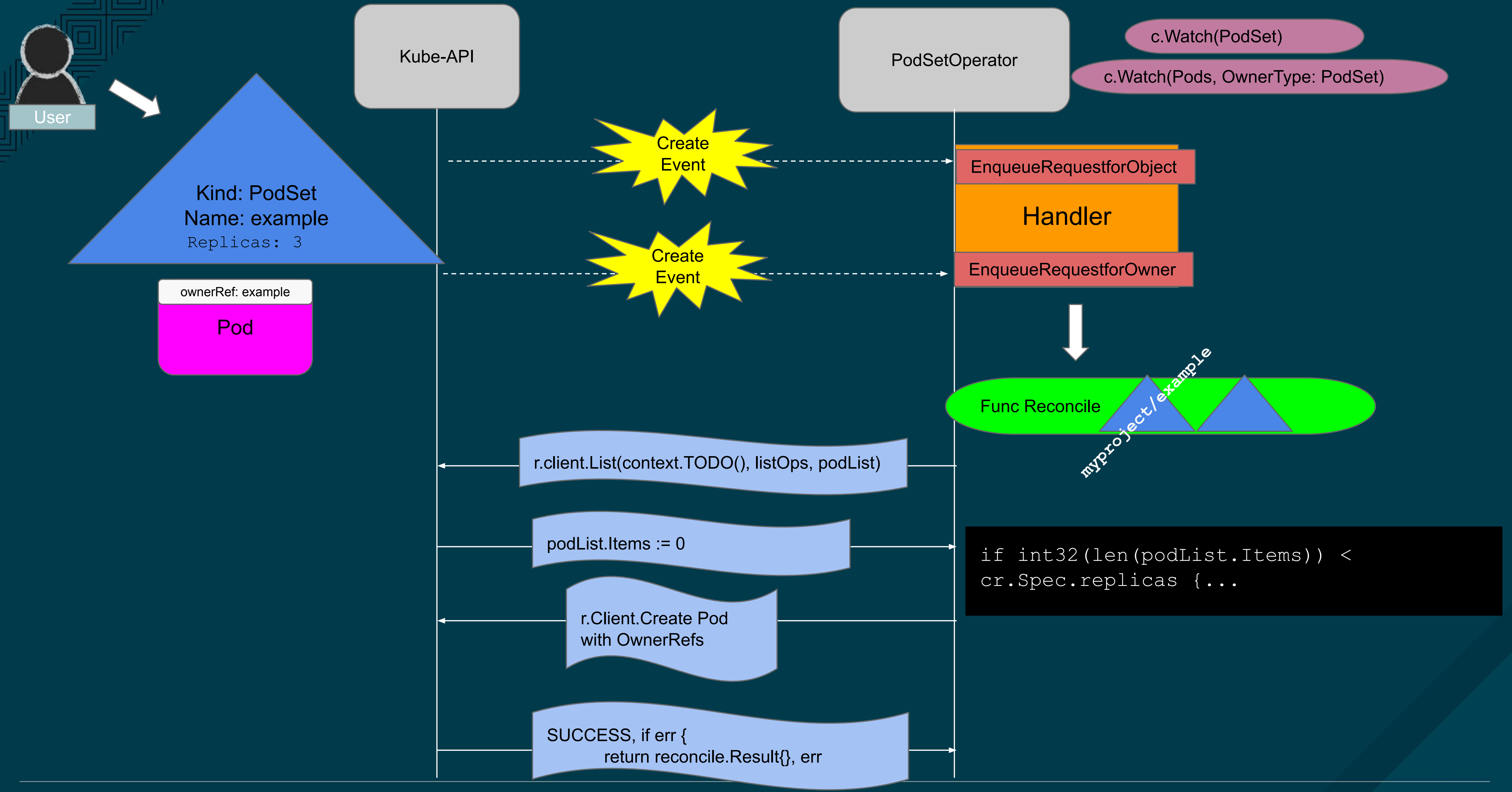


```
apiVersion: podset.redhat.com/v1alpha1
kind: PodSet
metadata:
  name: example
spec:
  replicas: 3
Status:
  availableReplicas: 3
  podNames:
  - “Pod1”
  ~~- “Pod2”~~  “Pod4”
  - “Pod3
```

A Pod Set Allows You to Scale Up/Down.



```
apiVersion: podset.redhat.com/v1alpha1
kind: PodSet
metadata:
  name: example
  namespace: default
spec:
  replicas:   1
status:
  podNames:
  ~~- "Pod1~~
  ~~- "Pod2"      - "Pod4"~~
  - "Pod3
```



User

Kind: PodSet
Name: example
Replicas: 3

ownerRef: example

Pod

Kube-API

PodSetOperator

c.Watch(PodSet)

c.Watch(Pods, OwnerType: PodSet)

Create Event

Create Event

EnqueueRequestforObject

Handler

EnqueueRequestforOwner

Func Reconcile

myproject/example

r.client.List(context.TODO(), listOps, podList)

podList.Items := 0

```
if int32(len(podList.Items)) <
cr.Spec.replicas {...
```

r.Client.Create Pod
with OwnerRefs

SUCCESS, if err {
return reconcile.Result{}, err

Red Hat