Kubernetes Operators

Over Easy

RHUG Edition

Ken Lee Sr. Specialist Solutions Architect

08/26/2020

Kubernetes Operators

Over Easy

Who am I?

Ken Lee

@keunlee

Red Hat - Solutions Architect - NA Central Region

What we’ll be discussing today

The Problem ...Which Birthed the Operator Pattern

What’s an Operator?

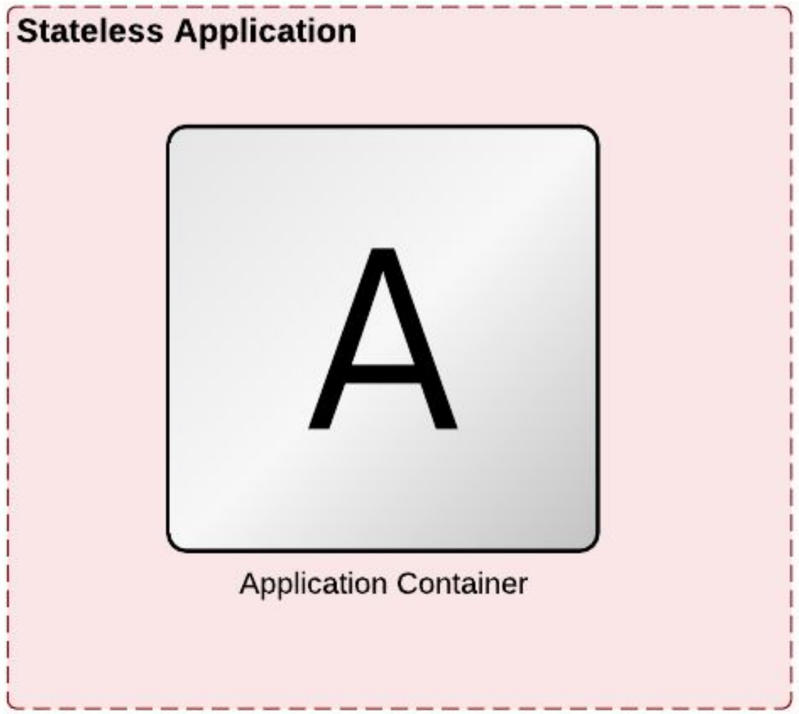
How does an Operator Work?

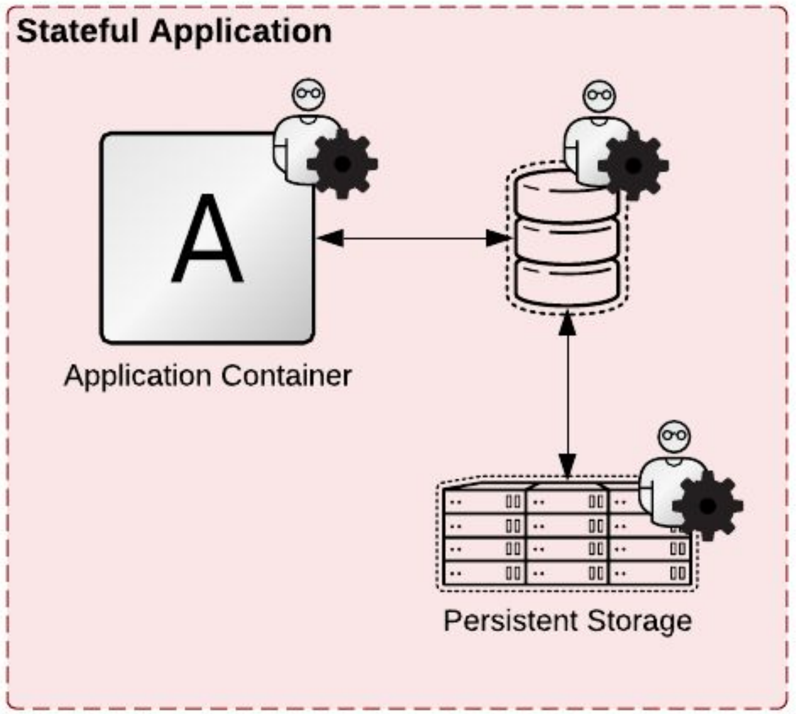
How do I make One?

Walkthrough

The Problem … Which Birthed the Operator Pattern

Stateless versus Stateful





How do you effectively automate Stateful applications on Kubernetes?

OPERATORS

What’s an Operator?

What’s an Operator?

"An operator is a Kubernetes controller that understands 2 domains: Kubernetes and something else. By combining knowledge of both domains, it can automate tasks that usually require a human operator that understands both domains."

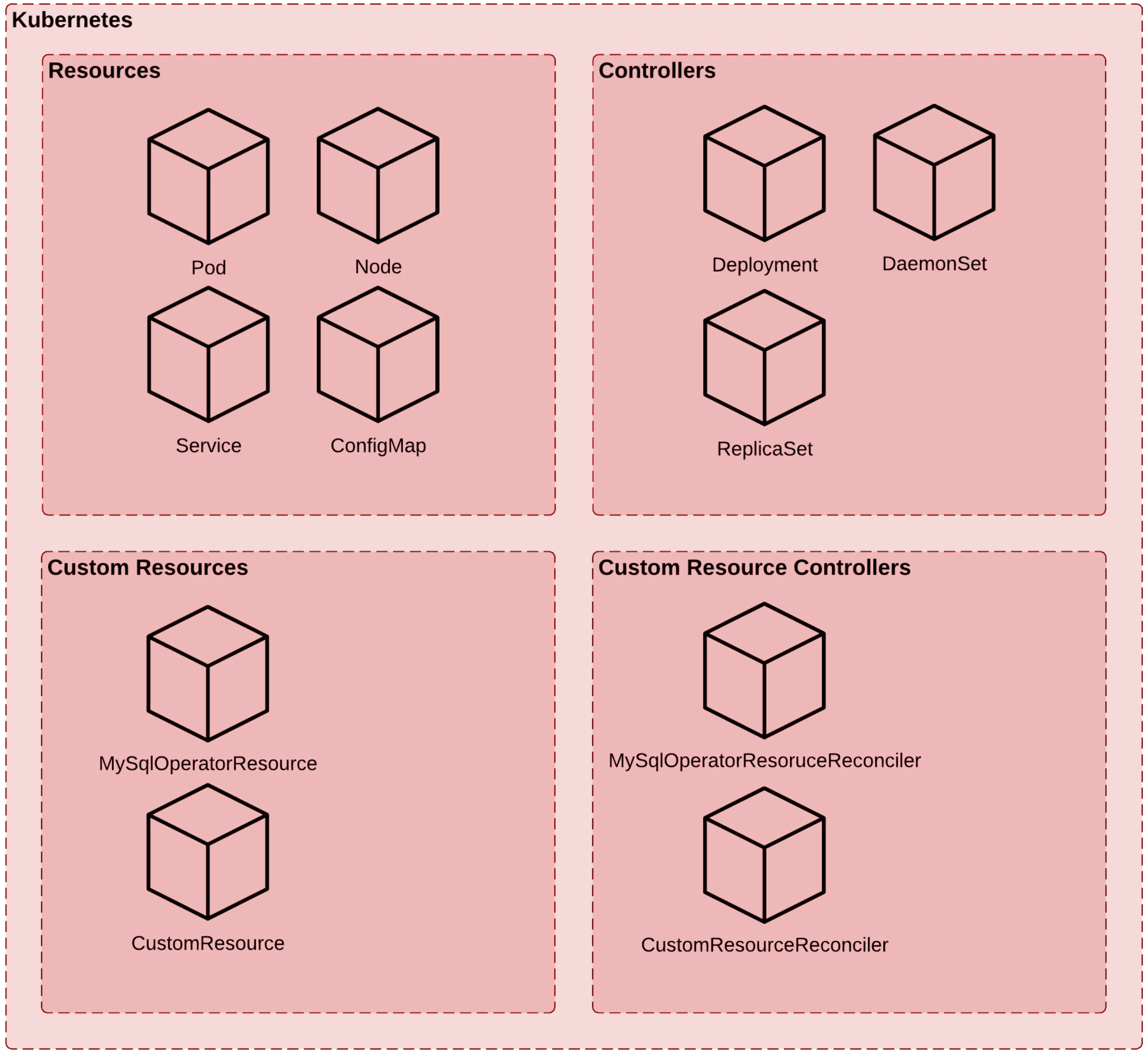
-Jimmy Zelinskie - Product and Engineering - CoreOS

https://bit.ly/3iS6AFx

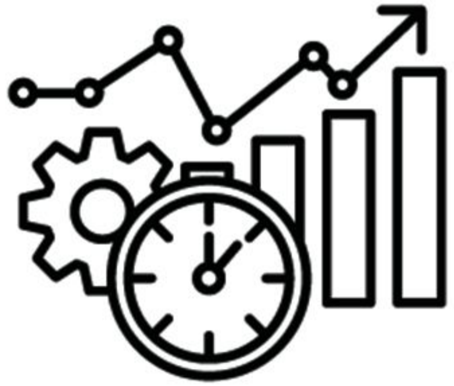
What’s an Operator?

Operator = Resource(s) + Controller(s) + Domain Specific Knowledge

Kubernetes: Resources + Controllers



Domain Specific Knowledge/Operations

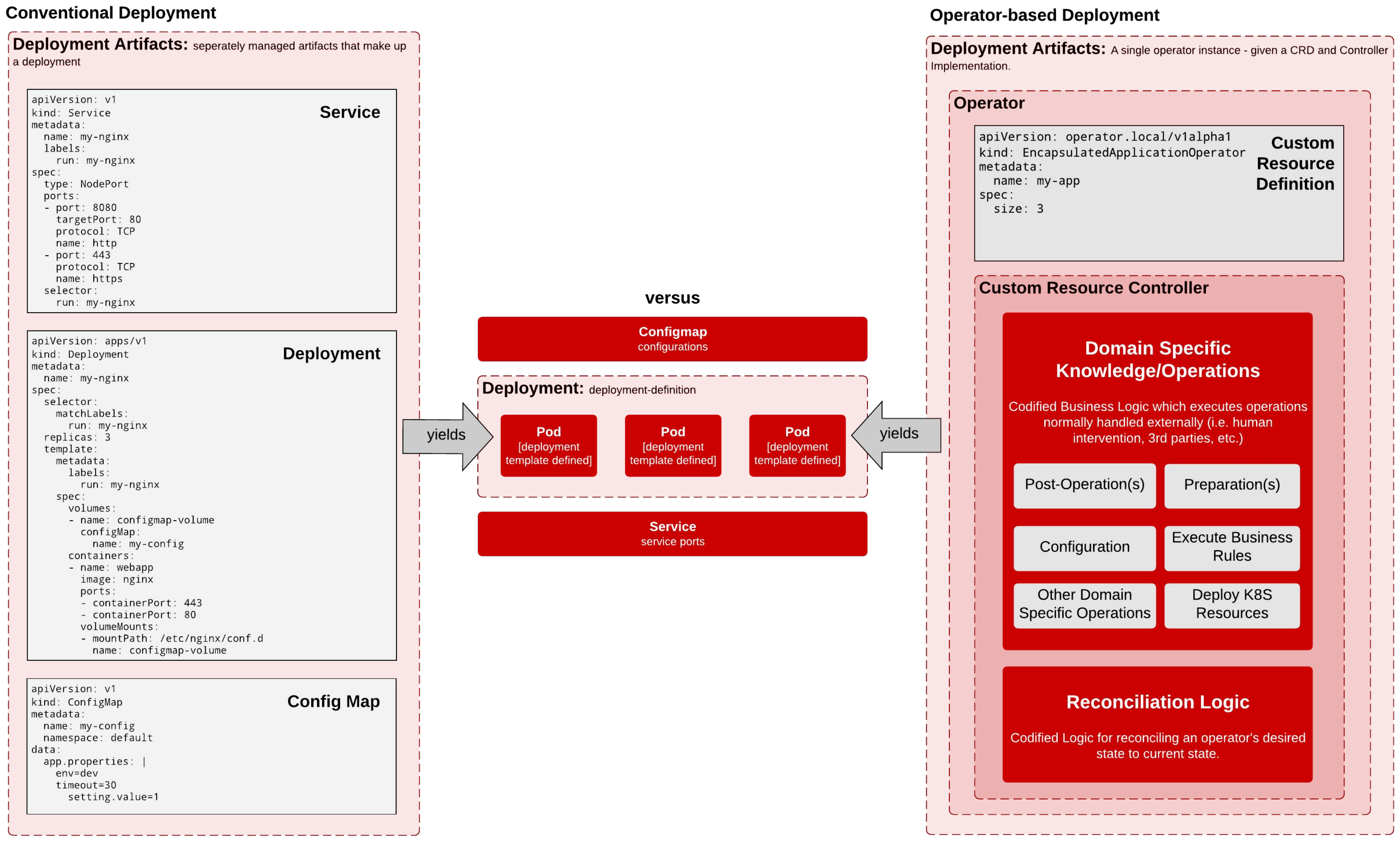


Examples of Domain Specific Knowledge/Operations (but not limited too)

- Fulfilling Configuration requirements
- Fulfilling Installation requirements
- Fulfilling Logging/Security requirements
- Fulfilling HA/Scaling requirements
- Application start-up and shutdown routines
- Process and workflow triggers
- Etc.

What's an Operator

Example: Conventional vs Operator based Deployments



Conventional Deployment

Deployment Artifacts: seperately managed artifacts that make up a deployment

Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: NodePort
  ports:
  - port: 8080
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    protocol: TCP
    name: https
  selector:
    run: my-nginx
```

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 3
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      volumes:
      - name: configmap-volume
        configMap:
          name: my-config
      containers:
      - name: webapp
        image: nginx
        ports:
        - containerPort: 443
        - containerPort: 80
        volumeMounts:
        - mountPath: /etc/nginx/conf.d
          name: configmap-volume
```

Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: default
data:
  app.properties: |
    env=dev
    timeout=30
      setting.value=1
```

Operator-based Deployment

Deployment Artifacts: A single operator instance - given a CRD and Controller Implementation.

Operator

Custom Resource Definition

```
apiVersion: operator.local/v1alpha1
kind: EncapsulatedApplicationOperator
metadata:
  name: my-app
spec:
  size: 3
```

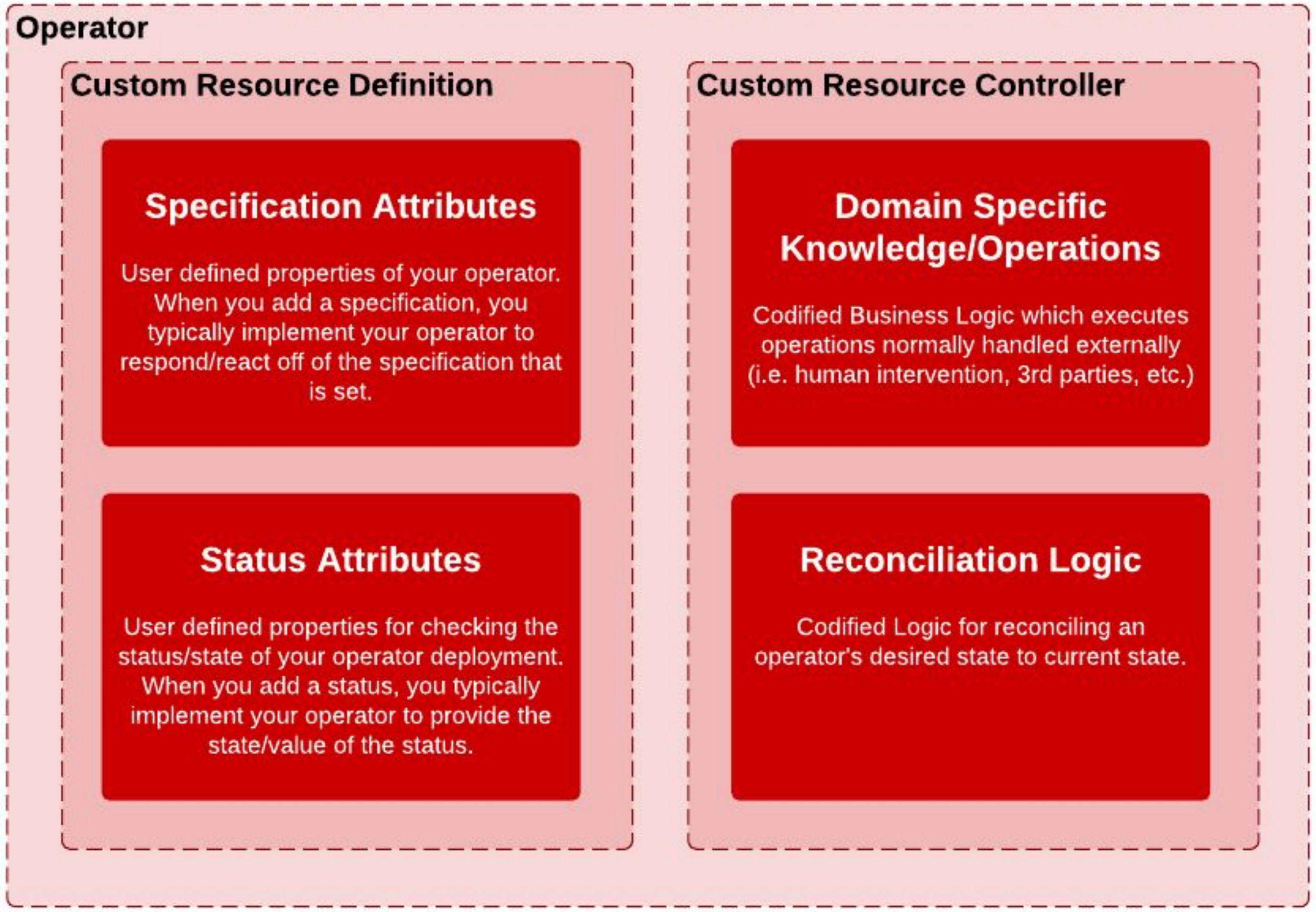
versus
Configmap
configurations
Deployment: deployment-definition
Pod [deployment template defined]
Pod [deployment template defined]
Pod [deployment template defined]
Service
service ports
yields
yields
Custom Resource Controller
Domain Specific Knowledge/Operations
Codified Business Logic which executes operations normally handled externally (i.e. human intervention, 3rd parties, etc.)
Post-Operation(s)
Preparation(s)
Configuration
Execute Business Rules
Other Domain Specific Operations
Deploy K8S Resources
Reconciliation Logic
Codified Logic for reconciling an operator's desired state to current state.

*** Operational specific tasks are carried out manually or potentially automated through other means.

How does an Operator work?

Operator Components

Operator = Resource(s) + Controller(s) + Domain Specific Knowledge



Specification Attributes

User defined properties of your operator. When you add a specification, you typically implement your operator to respond/react off of the specification that is set.

Status Attributes

User defined properties for checking the status/state of your operator deployment. When you add a status, you typically implement your operator to provide the state/value of the status.

Domain Specific Knowledge/Operations

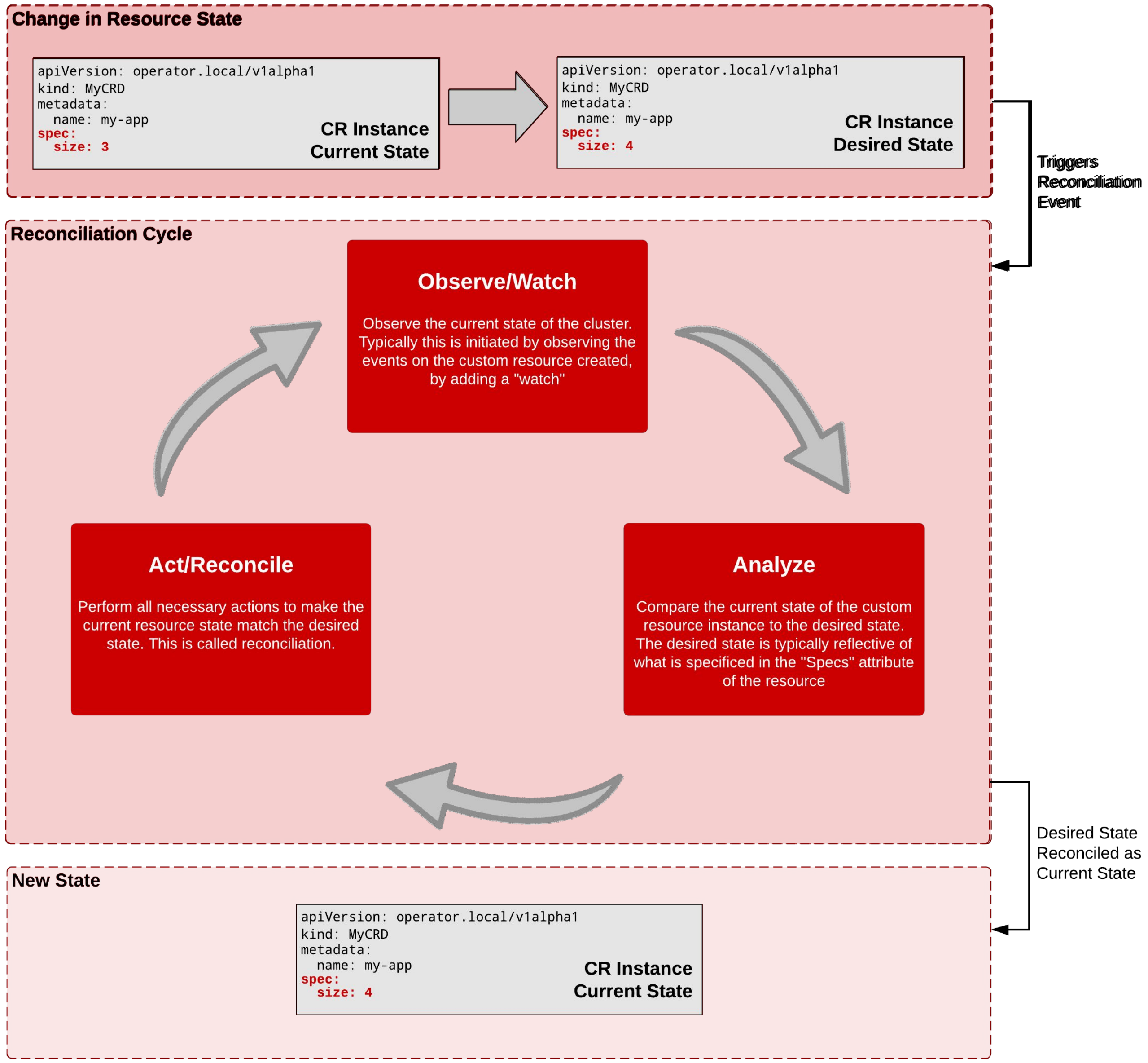
Codified Business Logic which executes operations normally handled externally (i.e. human intervention, 3rd parties, etc.)

Reconciliation Logic

Codified Logic for reconciling an operator's desired state to current state.

How does it work?

Custom Resource Controller - Reconciliation Cycle - Example


Change in Resource State
CR Instance Current State
CR Instance Desired State
Triggers Reconciliation Event
Reconciliation Cycle
Desired State Reconciled as Current State
New State
CR Instance Current State

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 3
```

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 4
```

Observe/Watch

Observe the current state of the cluster. Typically this is initiated by observing the events on the custom resource created, by adding a "watch"

Analyze

Compare the current state of the custom resource instance to the desired state. The desired state is typically reflective of what is specificed in the "Specs" attribute of the resource

Act/Reconcile

Perform all necessary actions to make the current resource state match the desired state. This is called reconciliation.

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 4
```

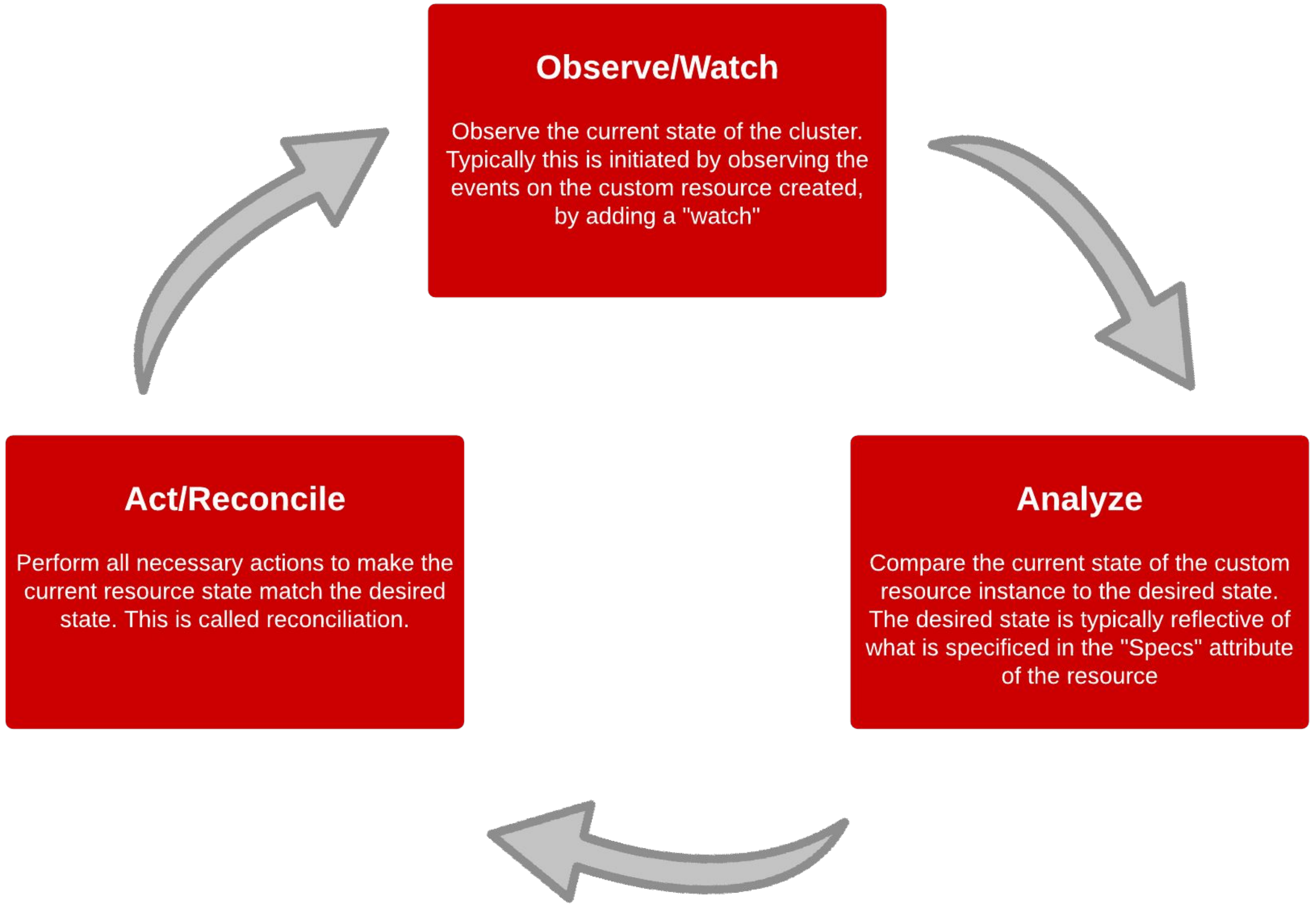
Red Hat

How do I make one?

Resources - Operator Frameworks and Libraries

- https://sdk.operatorframework.io/build/
  - Golang
  - Ansible
  - Helm
- https://book.kubebuilder.io/
  - Golang
- https://kudo.dev/
  - Yaml
- https://github.com/metacontroller/metacontroller
  - Jsonnet
- https://github.com/zalando-incubator/kopf
  - Python
- https://github.com/ContainerSolutions/java-operator-sdk
  - Java
- https://github.com/dot-i/k8s-operator-node
  - Typescript/NodeJS
- https://github.com/TremoloSecurity/kubernetes-javascript-operator
  - Javascript

Know the Reconciliation Cycle


Observe/Watch
Observe the current state of the cluster. Typically this is initiated by observing the events on the custom resource created, by adding a "watch"
Analyze
Compare the current state of the custom resource instance to the desired state. The desired state is typically reflective of what is specificed in the "Specs" attribute of the resource
Act/Reconcile
Perform all necessary actions to make the current resource state match the desired state. This is called reconciliation.

Resources - Learn more about Operators

- https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
- https://enterprisersproject.com/article/2019/2/kubernetes-operators-plain-english
- https://coreos.com/blog/introducing-operators.html
- https://www.openshift.com/blog/operator-framework-moves-to-cncf-for-incubation
- https://www.openshift.com/blog/kubernetes-operators-best-practices
- https://www.youtube.com/watch?v=8_DaCcRMp5I&t=3453s
- https://www.twitch.tv/videos/680494909
- https://github.com/keunlee/k8s-operators-over-ez

Walkthrough

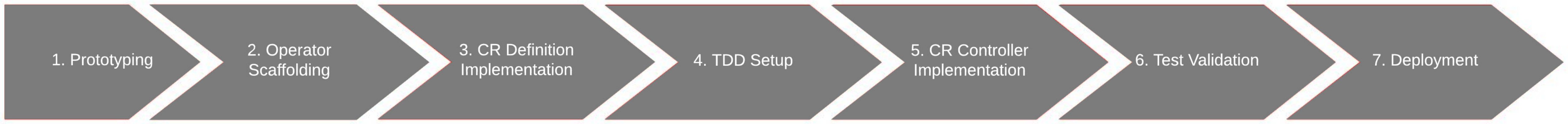
Operator Requirements (BDD Style)

TITLE: Overeasy Operator Requirements

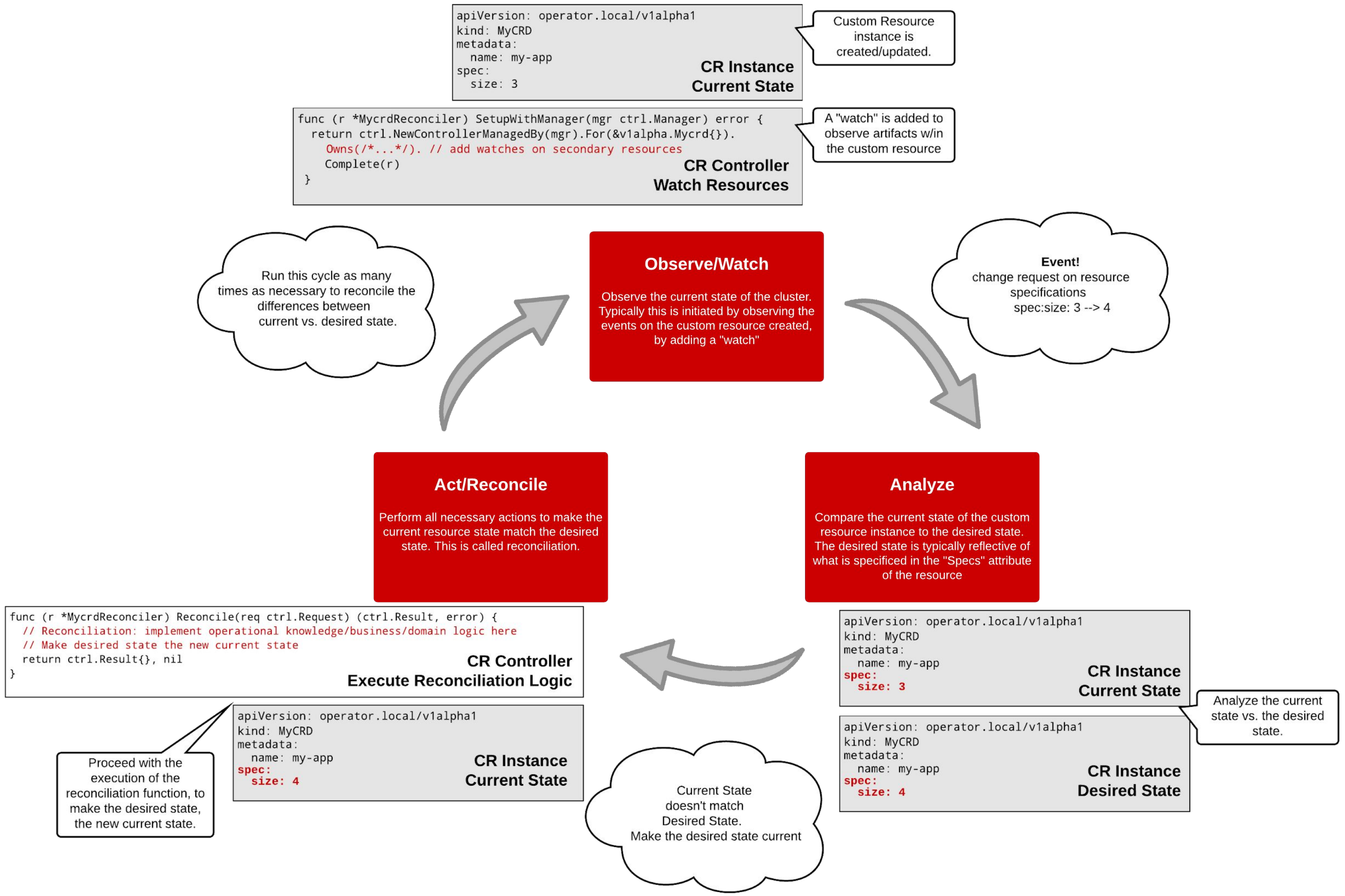
- **DESCRIPTION**
  - **AS A**: Developer
  - **I WANT**: An Operator with a single busybox pod that logs a user specified message and shuts down after a user specified amount of time. If a duration or message are not specified, then both will be supplied by a REST API call.
  - **SO THAT**: I can demonstrate the encapsulation of operational knowledge, leveraging the Operator Design Pattern.
- **SCENARIO 1**: Shutdown the busybox pod after a user specified amount of time in seconds
  - **GIVEN**: An Operator instance
  - **WHEN**: the specification `timeout` is set to a numeric value in seconds
  - **THEN**: the busy box pod will remain available for the specified `timeout` duration in seconds,
- **SCENARIO 2**: Log a user specified message before shutting down the busybox pod
  - **GIVEN**: An Operator instance
  - **WHEN**: the specification `message` is set to a string value
  - **THEN**: the busy box pod will log the message, from the `message` specification after the `timeout` duration has expired.
- **SCENARIO 3**: Retrieve the `timeout` and `message` from a given REST API if one and/or the other is not supplied.
  - **GIVEN**: An Operator instance
  - **WHEN**: the specification `message` OR `timeout` is NOT set
  - **THEN**: the busy box pod will supply these values from the following REST API: `GET http://my-json-server.typicode.com/keunlee/test-rest-repo/golang-lab00-response`
- **SCENARIO 4**: Update status `expired` and `logged` when the busybox pod has expired
  - **GIVEN**: An Operator instance
  - **WHEN**: the busy box pod's duration has expired
  - **AND**: the busy box pod has logged a message
  - **THEN**: set the operators `expired` status to `true`
  - **AND**: set the operators `logged` status to `true`

How do I make one?

Development Strategy



Reconciliation Cycle Revisited



```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 3
```

Custom Resource instance is created/updated.

```
func (r *MycrdReconciler) SetupWithManager(mgr ctrl.Manager) error {
  return ctrl.NewControllerManagedBy(mgr).For(&v1alpha.Mycrd{}).
    Owns(/*...*/). // add watches on secondary resources
    Complete(r)
 }
```

A "watch" is added to observe artifacts w/in the custom resource

Observe/Watch

Observe the current state of the cluster. Typically this is initiated by observing the events on the custom resource created, by adding a "watch"

Event!
change request on resource specifications
spec:size: 3 --> 4

Analyze

Compare the current state of the custom resource instance to the desired state. The desired state is typically reflective of what is specificed in the "Specs" attribute of the resource

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 3
```

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 4
```

Analyze the current state vs. the desired state.

Current State doesn't match Desired State.
Make the desired state current

Act/Reconcile

Perform all necessary actions to make the current resource state match the desired state. This is called reconciliation.

```
func (r *MycrdReconciler) Reconcile(req ctrl.Request) (ctrl.Result, error) {
  // Reconciliation: implement operational knowledge/business/domain logic here
  // Make desired state the new current state
  return ctrl.Result{}, nil
}
```

```
apiVersion: operator.local/v1alpha1
kind: MyCRD
metadata:
  name: my-app
spec:
  size: 4
```

Proceed with the execution of the reconciliation function, to make the desired state, the new current state.

Run this cycle as many times as necessary to reconcile the differences between current vs. desired state.

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat

Appendix

Additional Points of Interest

- What about Statefulsets? Can't I use those for managing and persisting State?
  - Short answer, YES
  - Think about what a Statefulset is. It's a resource controller too. The controller will manage the state of your pods with the use of persistent storage and a headless service.
  - Operators, offer a way for **you** to manage the state of your application, through **your** code.
- What about Helm Charts? When would you use a Chart vs an Operator?
  - We can try to use this as a general rule of thumb. If you need to codify operational knowledge of your K8S application as well as maintain state, then leveraging the Operator Pattern to facilitate the development of your K8S application, will serve you well.
  - However, if that's not the case, or the Operator pattern is just not your thing, you're not out of luck. You can still leverage constructs like Statefulsets to help you maintain state in your Kubernetes application, yet alone package a Statefulset configuration as part of your Helm Chart. The thing you have to keep in mind is how you manage and automate Domain Specific tasks and operations.