# Operators and Operator Framework

## Cloud Native Computing Meetup | August 2020

Simon Krenger

Technical Account Manager



Red Hat

# What we'll look at today

Introduction to Operators

Operator Framework

Operator Lifecycle Manager

OperatorHub

Examples & Best Practices

# So what is an Operator?

# Deploying things on Kubernetes is easy

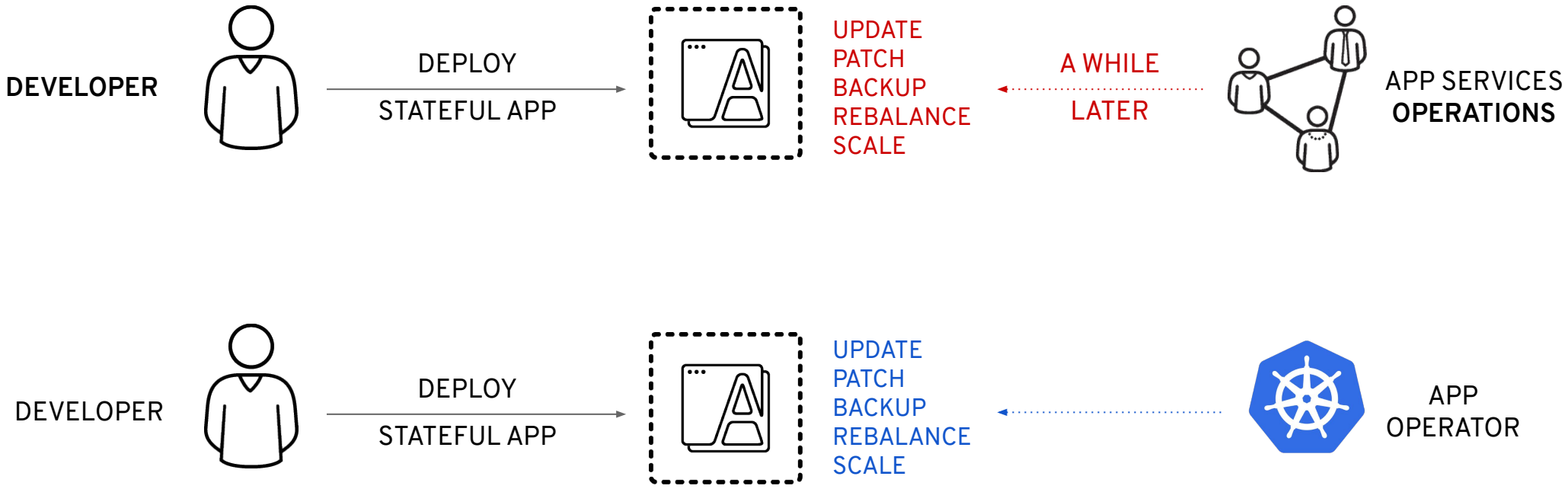
... but what about operating the thing?

At this point, deployment of an application on Kubernetes is a problem that is being solved by **Helm Charts, GitOps w/ ArgoCD, kustomize...**

But what about **application lifecycle, updates, backups, resizing, recovery, monitoring, scaling, tuning, configuration changes?**

An Operator is a **Site Reliability Engineer** implemented in software in a **Kubernetes-native** way

# What does an Operator do?



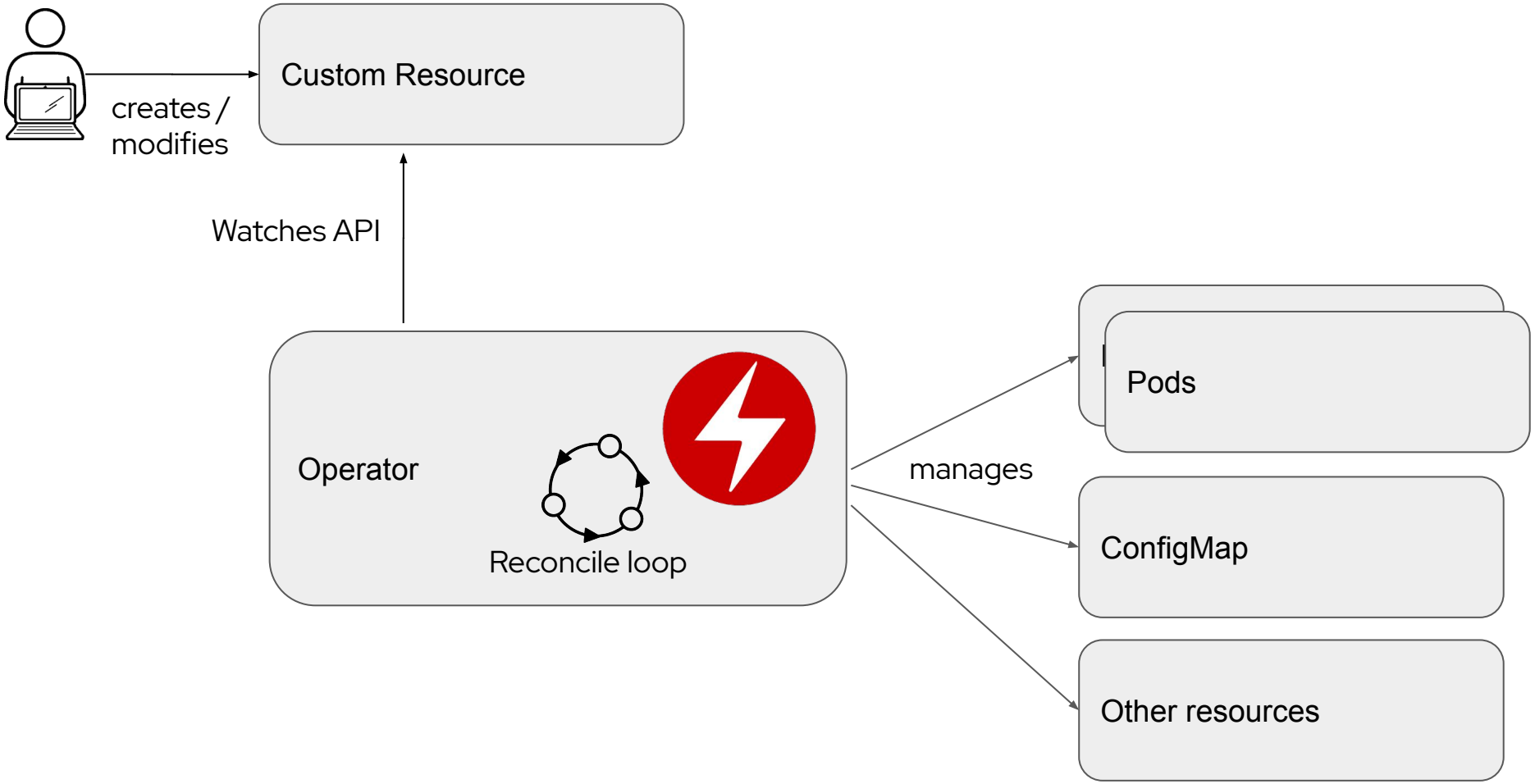
# Operator Pattern

Operators are software **extensions to Kubernetes** that make use of **custom resources** to manage applications and their components. Operators follow Kubernetes principles, notably the **control loop**.

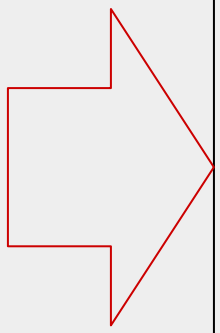
Basically an Operator is a **custom Kubernetes controller for an application**



Source:
https://kubernetes.io/docs/concepts/extend-kubernetes/operator/


Custom Resource
creates / modifies
Watches API
Operator
Reconcile loop
manages
Pods
ConfigMap
Other resources

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 2.4.1
    replicas: 3
    storage:
      type: ephemeral
    listeners:
      plain: {}
      tls: {}
      external:
        type: nodeport
        tls: false
  zookeeper:
    replicas: 1
    storage:
      type: ephemeral
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

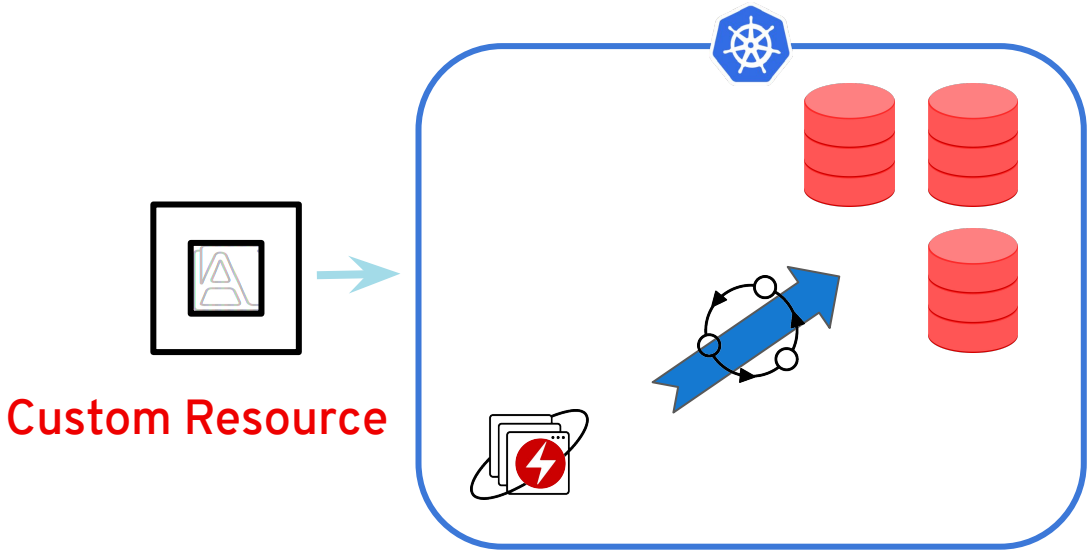


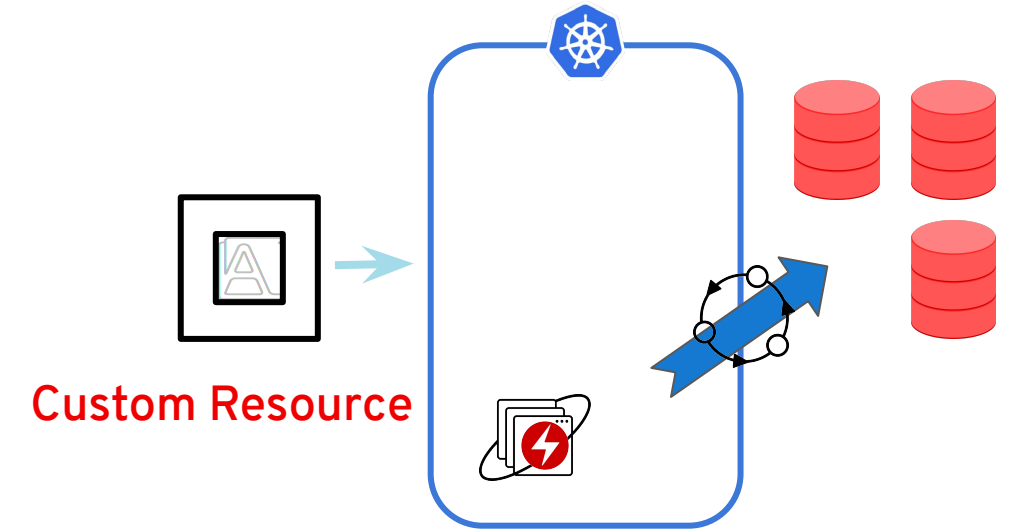
```
kubectl get kafka,deployment,pods,statefulset,service,cm

NAME                                DESIRED KAFKA REPLICAS   DESIRED ZK REPLICAS
kafka.kafka.strimzi.io/my-cluster   3                        1

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-cluster-entity-operator    1/1     1            1           25m

NAME                                             READY   STATUS    RESTARTS   AGE
pod/my-cluster-entity-operator-7686d45df9-2h86b  3/3     Running   0          25m
pod/my-cluster-kafka-0                           2/2     Running   0          26m
pod/my-cluster-kafka-1                           2/2     Running   0          26m
pod/my-cluster-kafka-2                           2/2     Running   0          26m
pod/my-cluster-zookeeper-0                       1/1     Running   0          26m

NAME                                    READY   AGE
statefulset.apps/my-cluster-kafka       3/3     26m
statefulset.apps/my-cluster-zookeeper   1/1     26m

NAME                                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
service/my-cluster-kafka-0                  NodePort    172.30.102.153   <none>        9094:31347/TCP               26m
service/my-cluster-kafka-1                  NodePort    172.30.183.107   <none>        9094:31427/TCP               26m
service/my-cluster-kafka-2                  NodePort    172.30.187.202   <none>        9094:31547/TCP               26m
service/my-cluster-kafka-bootstrap          ClusterIP   172.30.184.106   <none>        9091/TCP,9092/TCP,9093/TCP   26m
service/my-cluster-kafka-brokers            ClusterIP   None             <none>        9091/TCP,9092/TCP,9093/TCP   26m
service/my-cluster-kafka-external-bootstrap NodePort    172.30.91.41     <none>        9094:30693/TCP               26m
service/my-cluster-zookeeper-client         ClusterIP   172.30.200.221   <none>        2181/TCP                     26m
service/my-cluster-zookeeper-nodes          ClusterIP   None             <none>        2181/TCP,2888/TCP,3888/TCP   26m

NAME                                              DATA   AGE
configmap/my-cluster-entity-topic-operator-config 1      25m
configmap/my-cluster-entity-user-operator-config  1      25m
configmap/my-cluster-kafka-config                 3      26m
configmap/my-cluster-zookeeper-config             2      26m
```

# Demo

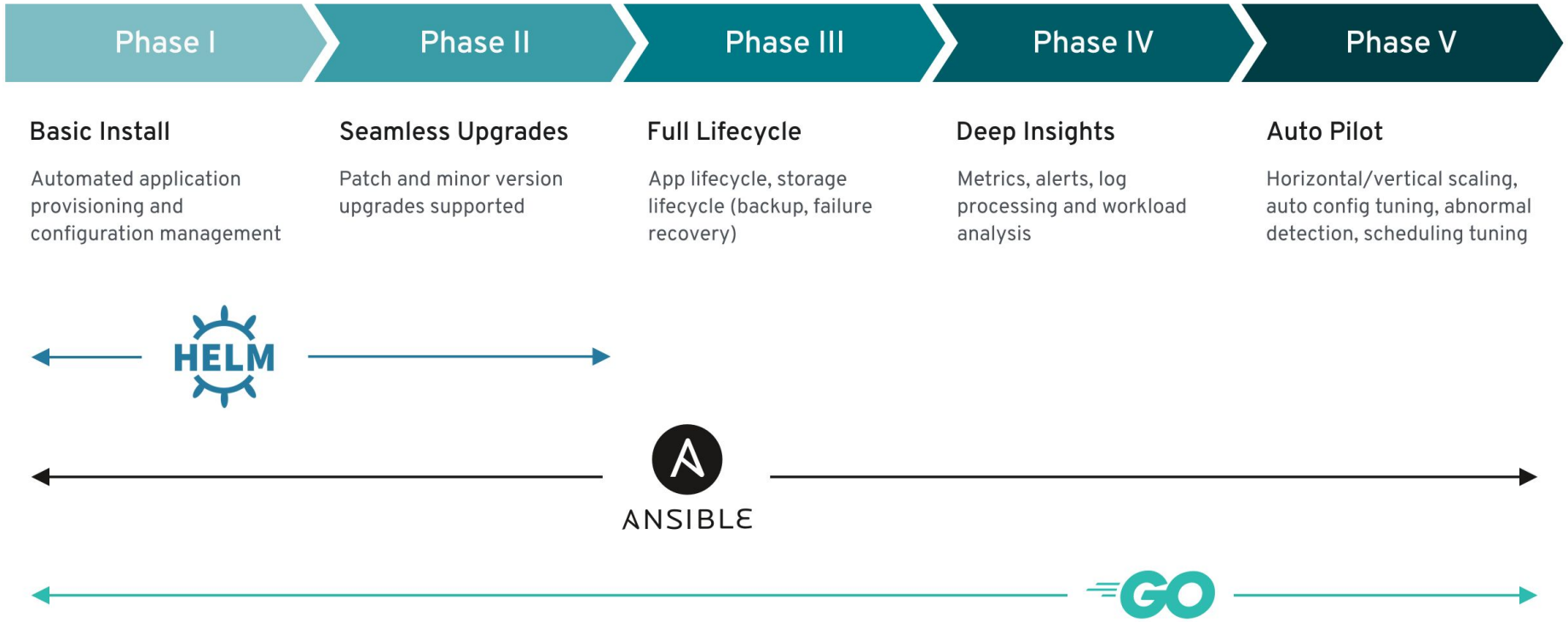
# Patterns

## Operators can also manage cluster-external resources



# Types of Operators



# Some practical tips to interact with Operators

- `kubectl api-resources`
  `kubectl explain --recursive <crd>`

  to show CustomResources and explain fields

- Primary `.Status` fields on Operator CRDs to check the status of a resource:

  - Available
  - Progressing
  - Degraded

- The `app.kubernetes.io/managed-by` label on Pods typically tells you which Operator is responsible for a certain Pod

# Operator Framework

# What is the Operator Framework?

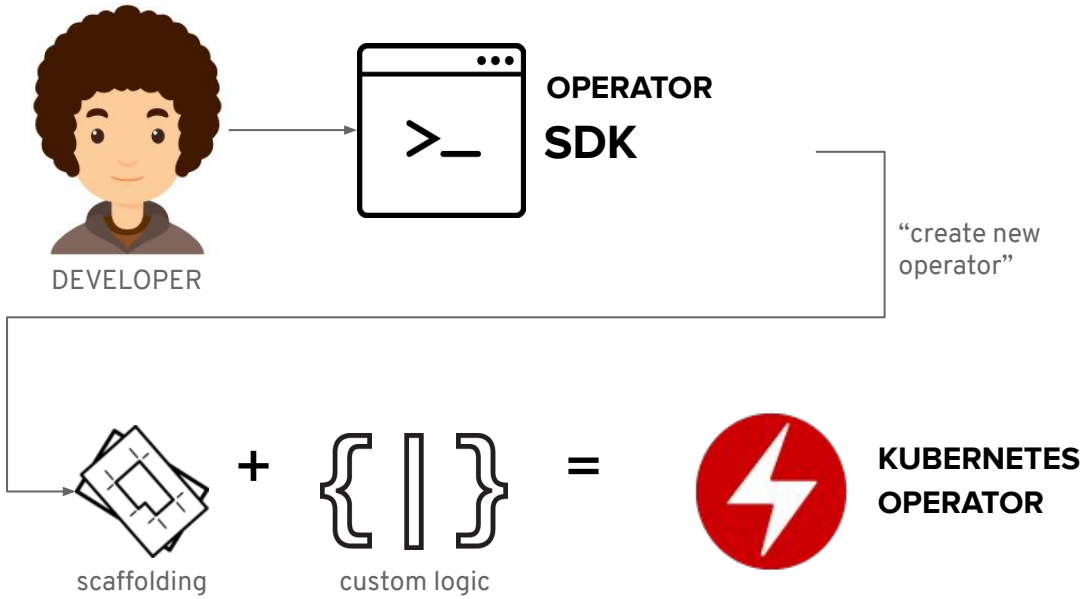
The **Operator Framework is a CNCF incubator project** and toolkit to manage Operators in an effective, automated and scalable way

- Operator SDKs for Go, Ansible, Helm
- Operator Lifecycle Manager
- Operator Registry
- Community Operators



Sour ce:
https://github.com/operator-framework/operator-lifecycle-manager
https://landscape.cncf.io/selected=operator-framework

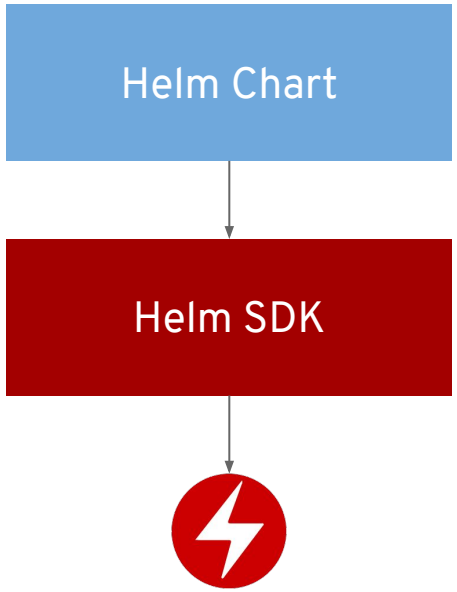
# Operator SDK in Action



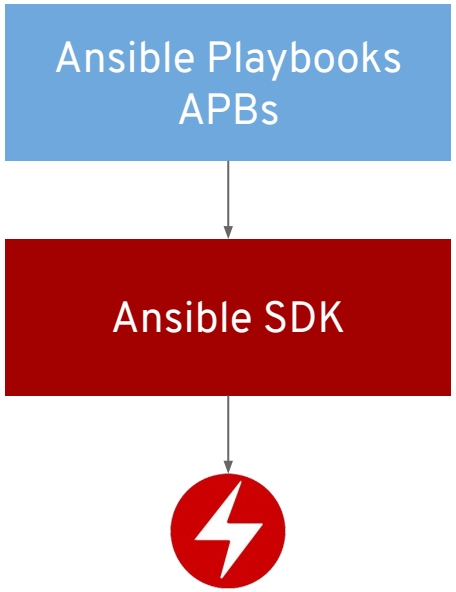

*implemented as a container image plus:*

```
Deployment
Role
ClusterRole
RoleBinding
ClusterRoleBinding
ServiceAccount
CustomResourceDefinition
```

# Operator SDK Patterns




Build operators from Helm chart, without any coding



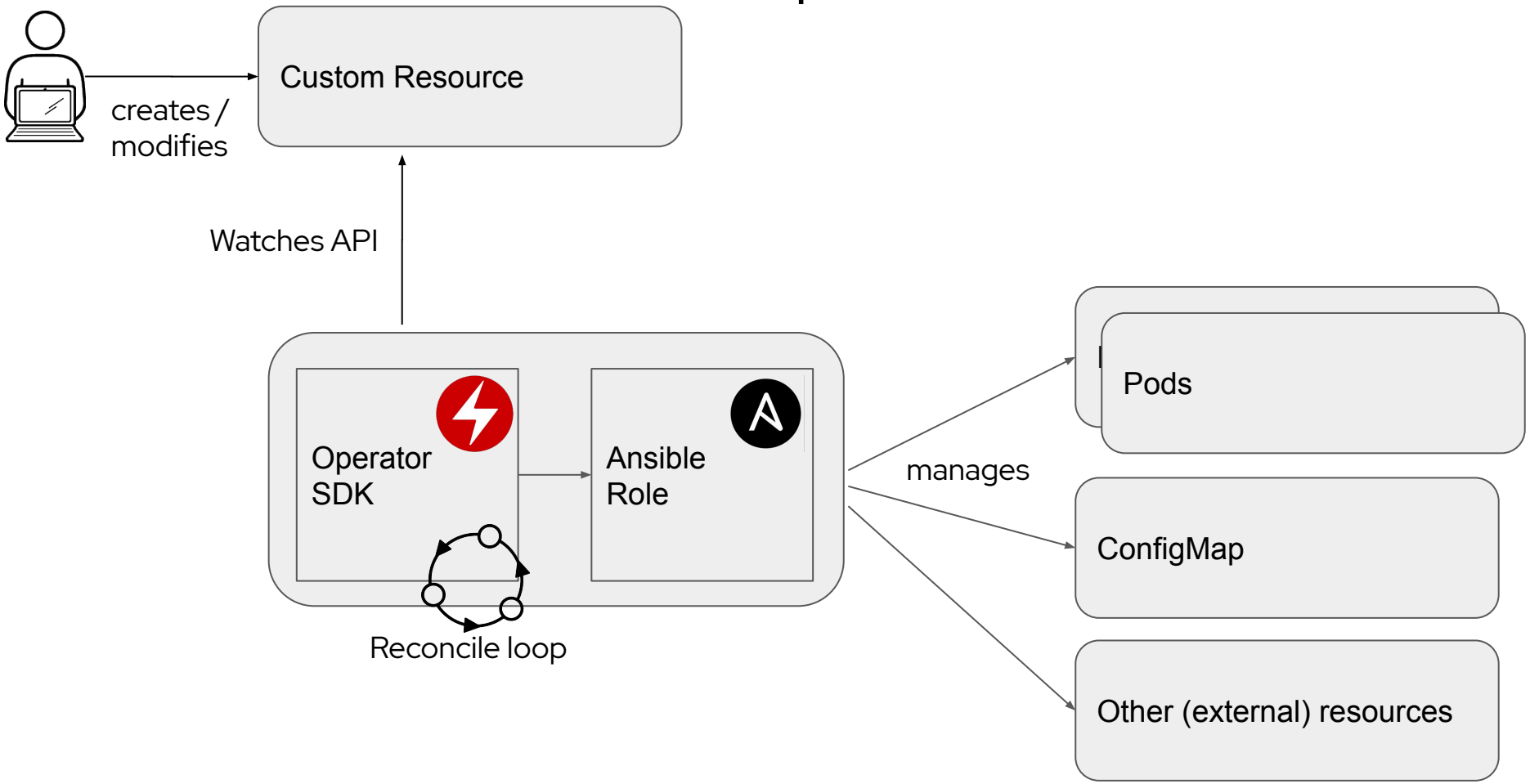

Build operators from Ansible playbooks and APBs


Build advanced operators for full lifecycle management



Source:
https://sdk.operatorframework.io/docs/building-operators/

# Ansible Operator



# Demo

# Operator Lifecycle Manager (OLM)

# Operator Lifecycle Manager

The Operator Lifecycle Manager extends Kubernetes to provide a declarative way to **install, manage, and upgrade Operators and their dependencies** in a cluster.

OLM allows a user to “subscribe” to an Operator, which unifies installation and updates in a single concept.

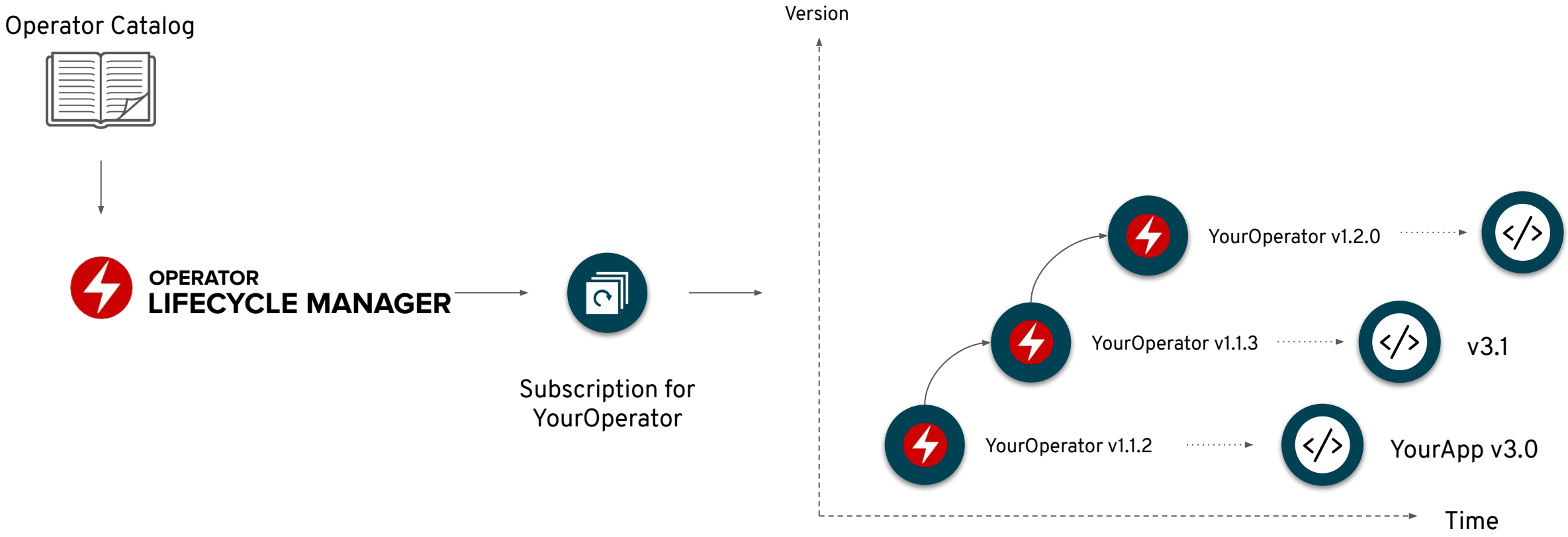


Sour ce:
https://github.com/operator-framework/operator-lifecycle-manager

# Using OLM to install Operators


Operator Deployment
Custom Resource Definitions
RBAC
API Dependencies
Update Path
Metadata

# Using OLM to update Operators



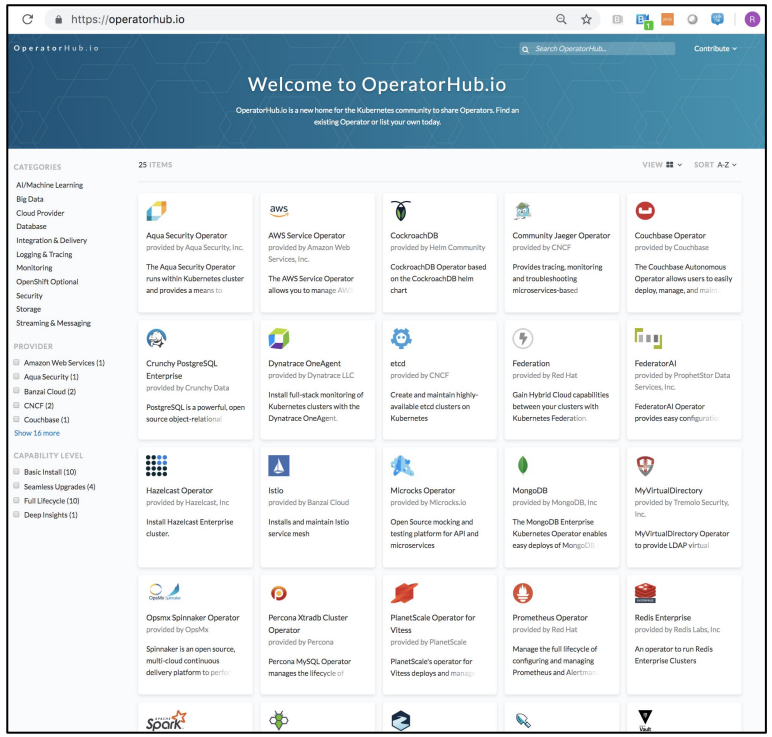



Sour ce:
https://github.com/operator-framework/operator-lifecycle-manager
https://github.com/operator-framework/operator-registry

# Demo

# OperatorHub.io

## A place to share Operators




The public registry for finding Kubernetes Operator backed services. More than 140 Community Operators available

Popular Operators:

- Strimzi (Kafka)
- Prometheus
- Istio
- Crunchy PostgreSQL



Source:
https://operatorhub.io
https://github.com/operator-framework/community-operators

# Operator Use Cases

Use Cases **ideal** for operator development:

- Stateful applications
- Clustered or high-availability applications
- Multiservice applications
- Microservices

Use Cases **less ideal** for operator development:

- Stateless applications
- Infrastructure / Host agents



Source:
https://redhat-connect.gitbook.io/certified-operator-guide/

# Real-world examples for Operators

## Customer Use Cases

- Crunchy Postgres Operator for databases in the development environment
- Creating a complete development environment / pipelines with a single CR (Ansible Operator)
- Configuration of an external firewall with k8s objects (Ansible Operator)

**OpenShift 4 components** are all managed by Operators

- ClusterVersionOperator
- MachineOperator
- etcd
- API Server, Controller Manager
- Ingress
- ... and around 25 more “Cluster Operators”

# Operator Best Practices

## Development

- **One Operator per managed application**
- Write an Operator-of-Operators for complex, multi-tier application stacks
- CRD can only be owned by a single Operator, shared CRDs should be owned by a separate Operator
- One controller per custom resource definition
- **Use an SDK like Operator SDK**
- Make sure the Operator Reconciliation loop does not spam the API
- Do not hard-code namespaces or resources names
- Make watch namespace configurable
- Use OpenAPI spec on CRDs

# Operator Best Practices

## Running Operators

- Does not run as root
- Does not self-register CRDs
- Writes **meaningful status information** on Custom Resource objects
- Is capable of updating from a previous version of the Operator
- Is capable of managing an Operand from an older Operator version
- Uses CRD conversion (webhooks) if API/CRDs change
- Uses Admission Webhooks to reject invalid CRs
- Should always be able to **deploy and come up without user input**
- Offers configuration via an "**Configuration CR**"

# 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

# Additional learning Resources

- Demos: https://github.com/simonkrenger/cnc-operators-demos
- CNCF Incubating projects: https://www.cncf.io/projects/
- Hands-on labs: https://www.katacoda.com/openshift/courses/operatorframework
- Operator SDK: https://sdk.operatorframework.io
- OperatorHub: https://operatorhub.io/