

Operate Kubernetes workloads ...

... extend the platform with the ***operator pattern!***

Paolo Patierno

Principal Software Engineer

Who am I?

apiVersion: v1

kind: PrincipalSoftwareEngineer

metadata:

name: Paolo Patierno

namespace: Red Hat, Messaging & Data Streaming

labels:

cncf/maintainer: Strimzi

eclipse/commmitter: Vert.x, Hono & Paho

microsoft/mvp: Azure

annotations:

family: dad of two, husband of one

sports: running, swimming, motogp, vr46, formula1, ferrari, ssc

napoli

community: cncf napoli, devday

spec:

replicas: 1

containers:

- image: patiernohub.io/paolo:latest



@ppatierno

Kubernetes

" A system for ..."

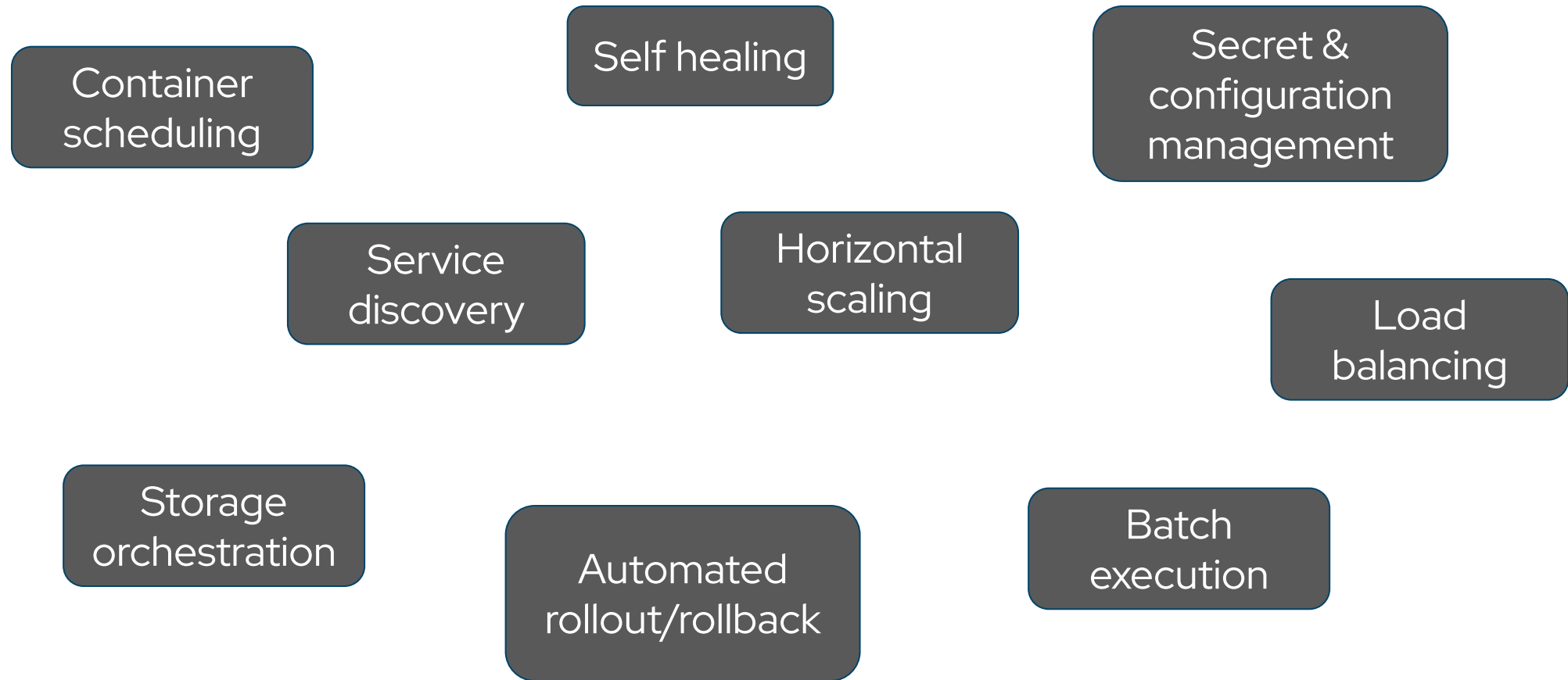
" ... automating deployment ..."

" ... scaling ..."

" ... management ..."

" ... of containerized applications ..."

" It's like a Linux kernel ... but for distributed systems"



Workloads





How does Kubernetes handle scaling, rollout, batch execution and so on?



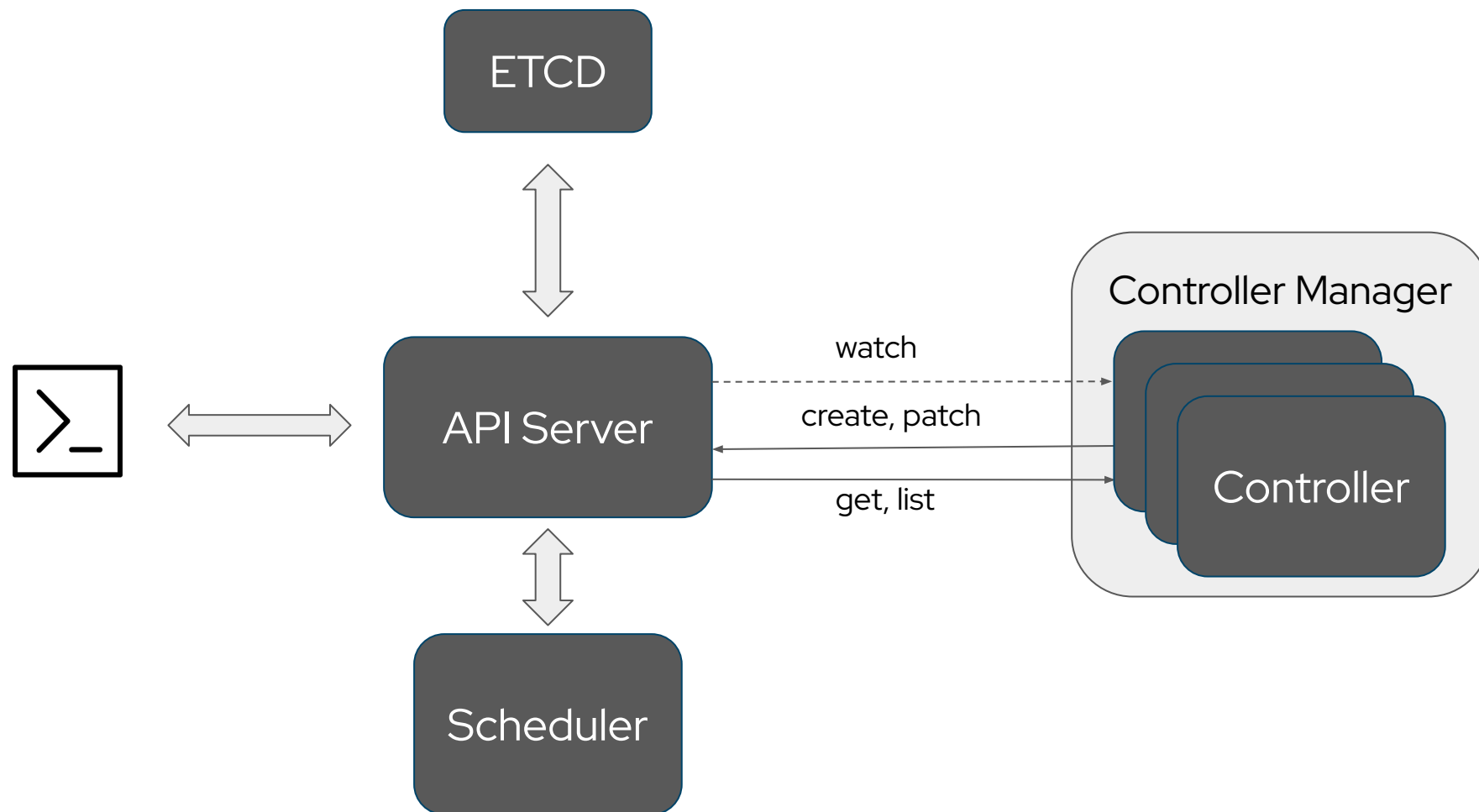
Workloads

- ▶ Don't use Pod(s) ... let's use something more sophisticated!
- ▶ ReplicaSet
 - Guarantees a specific number of running replicas
 - Spins pods, based on a template, if there are not enough ...
 - ... or deletes pods if too many
- ▶ Deployment
 - It's based on ReplicaSet (used to run replicas)
 - Adds extra layer for rollout and rollback
- ▶ ... and more with StatefulSet, Job, DaemonSet ...



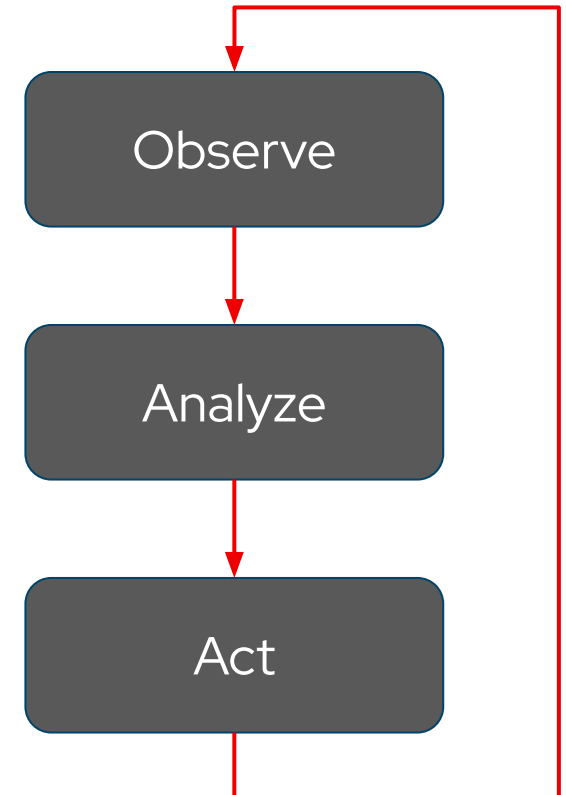
How does it work?
Let's use a controller!!!
.... But not this one ;-)





Reconcile Loop

- ▶ Observe
 - Watch for resource/object creation or changes
- ▶ Analyze
 - Check that the resource/object desired state ("spec") reflects the current state on the cluster
- ▶ Act
 - Makes the needed changes



```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-application
          image: quay.io/ppatierno/my-application:latest
```

my-replicaset-bf5zv

my-replicaset-1tf5a

my-replicaset-gb65f

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-application
        image: quay.io/ppatierno/my-application:latest
```

replicas: 5

my-replicaset-bf5zv

my-replicaset-1tf5a

my-replicaset-gb65f

my-replicaset-5tfgs

my-replicaset-rf43g

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-1
  labels:
    app: my-app
spec:
  containers:
    - name: my-app
      image: quay.

apiVersion: v1
kind: Pod
metadata:
  name: my-pod-2
  labels:
    app: my-app
spec:
  containers:
    - name: my-application
      image: quay.io/ppatierno/my-application:latest
```

my-replicaset-bf5zv

my-replicaset-1tf5a

my-replicaset-gb65f

~~my-pod-1~~~~my-pod-2~~

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-1
  labels:
    app: my-app
spec:
  containers:
    - name: my-app
      image: quay.

apiVersion: v1
kind: Pod
metadata:
  name: my-pod-2
  labels:
    app: my-app
spec:
  containers:
    - name: my-application
      image: quay.io/ppatierno/my-application:latest
```

my-pod-1

my-pod-2

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-application
          image: quay.io/ppatierno/my-application:latest
```

my-pod-1

my-pod-2

my-replicaset-65rt3

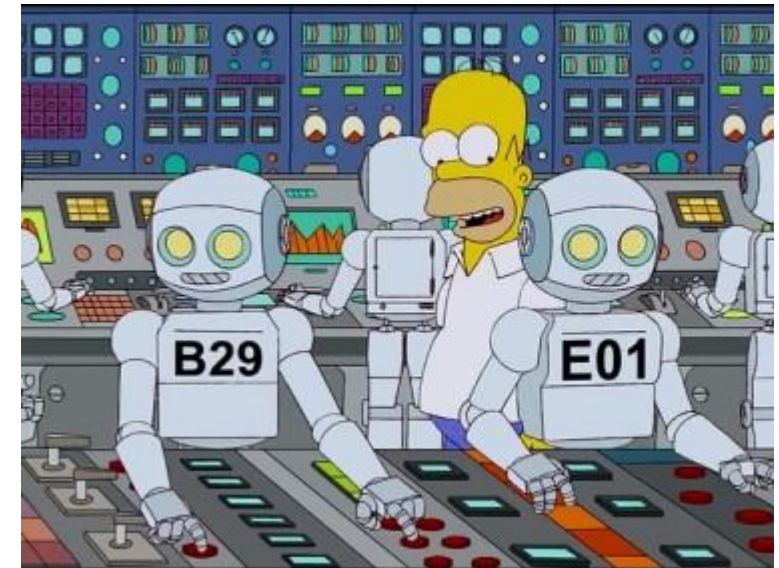


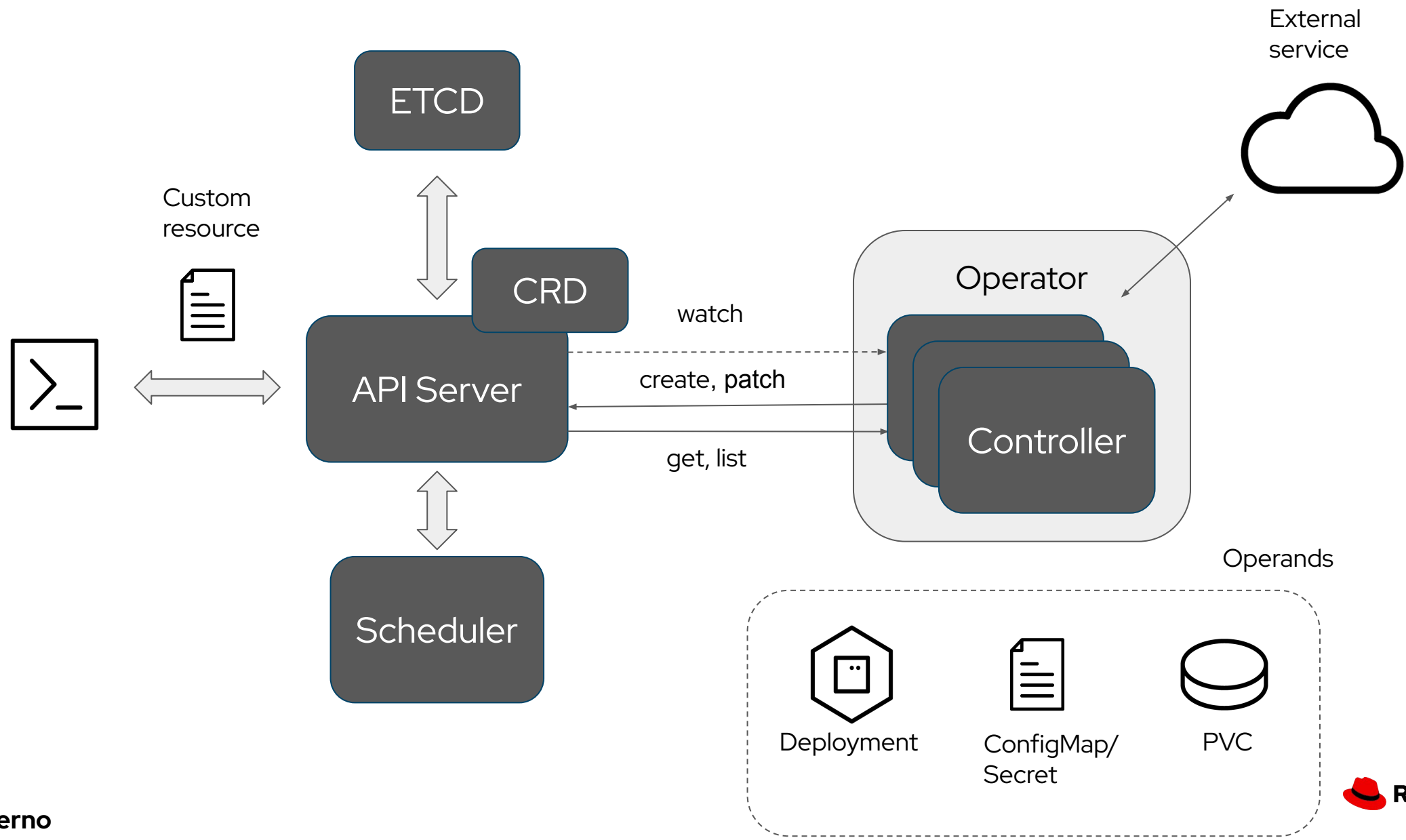
How to automate operating complex applications? The Operator pattern!



Operator

- ▶ It's yet another containerized application!
- ▶ Has the knowledge of a specific business domain
- ▶ Manage the application lifecycle
- ▶ Leverage CRDs (Custom Resource Definition) to extend API server
- ▶ Takes care of one (or more) custom resources/objects
 - By having a controller for each resource
 - Creating native Kubernetes resources ... aka "operands"
 - Leveraging built-in controllers via API server interaction





Operator

- ▶ Each “internal” controller watches a corresponding custom resource
 - Create/update/delete the corresponding “operands” as native Kubernetes resources but ... could be other custom resources for other operators :-)
- ▶ Watch the “operands”
 - They can be touched only by operator controllers ... not humans
 - Reverts back any manual changes
 - “Owned by” the custom resource, leveraging Kubernetes garbage collection
- ▶ Can interact with external service
 - A custom resource could be related to handle a non-Kubernetes service (i.e. Azure Service Operator, ...)



STRIMZI

THE Apache Kafka operator

- ▶ Open source project licensed under Apache License 2.0
- ▶ Focuses on running Apache Kafka on Kubernetes
 - Container images for Apache Kafka, Apache ZooKeeper and other components
 - Operators for deploying, managing and configuring Kafka clusters
- ▶ Provides a Kubernetes-native experience
 - Not only Kafka clusters, but also users, topics and the rest of Kafka ecosystem
- ▶ CNCF sandbox project since September 2019

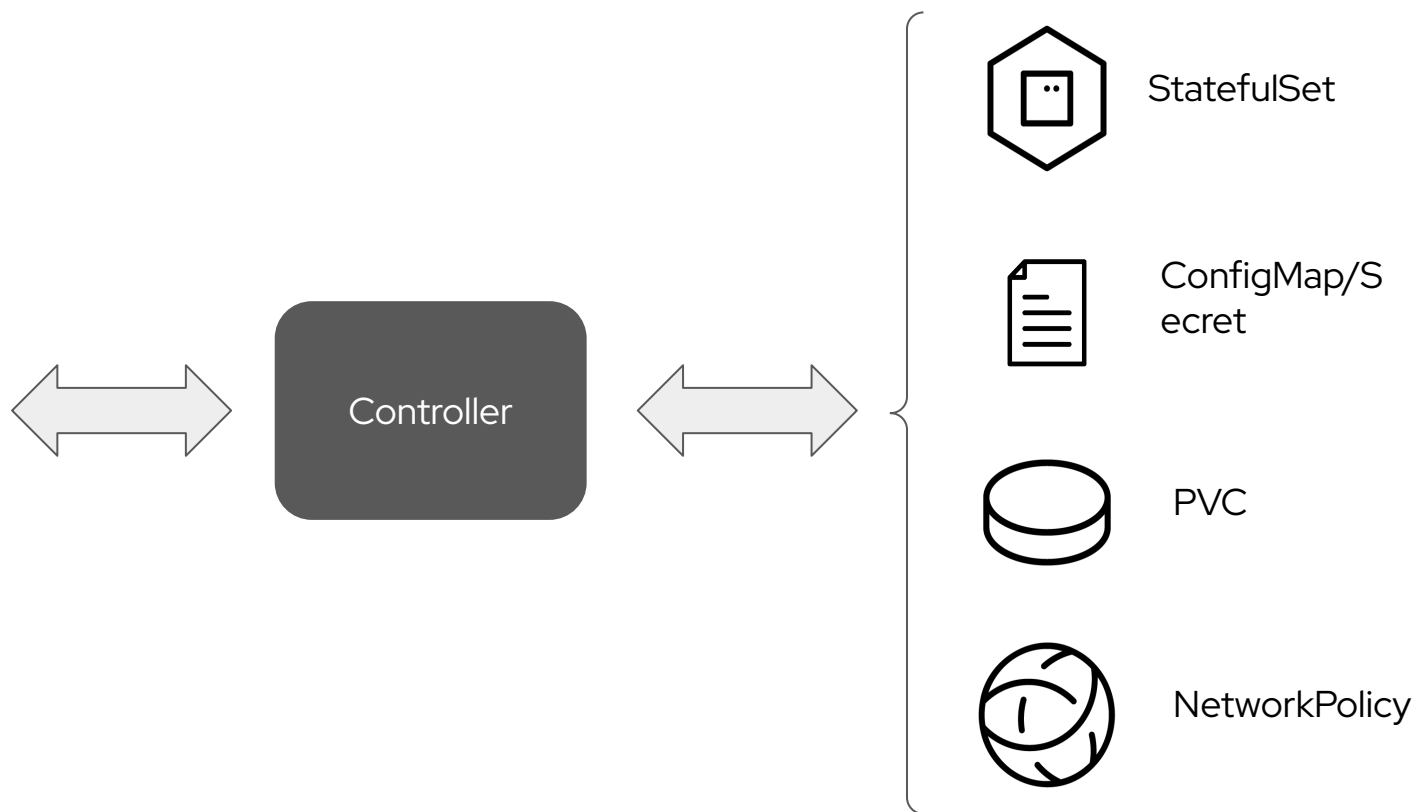
From the Custom Resource Definition ...

- ▶ It's a ... Kubernetes resource!
- ▶ Declare a new Kubernetes "kind"
 - Group
 - versions
- ▶ Define the new "kind" structure using an OpenAPI schema
 - Spec
 - status

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: kafkas.kafka.strimzi.io
spec:
  group: kafka.strimzi.io
  names:
    kind: Kafka
    listKind: KafkaList
  ...
  versions:
  - name: v1beta2
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            # spec definition with for
            # the custom resource
            ...
          status:
            # status definition reported back
            # in the custom resource
```

... to the Custom Resource

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 3.1.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      default.replication.factor: 3
      min.insync.replicas: 2
      inter.broker.protocol.version: "3.1"
      ...
    storage:
      type: ephemeral
  zookeeper:
    replicas: 3
    storage:
      type: ephemeral
status:
  ...
  ...
```

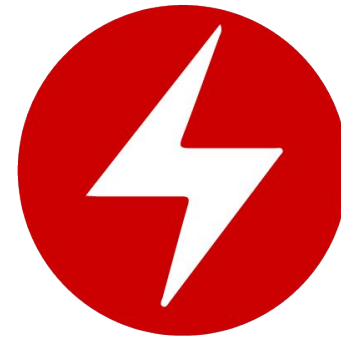




Why? What about Helm Charts?



VS



Helm Charts

- ▶ “Package manager” for Kubernetes
 - Charts = template + values
- ▶ Rely on Kubernetes built-in resources (i.e. Deployment, ConfigMap, ...)
- ▶ Simplify to write YAMLs with parameters via templating
- ▶ Simplify day-1 operation, for deploying applications
- ▶ Key problems fixed
 - Deploy same application with different configuration
 - Deploy same application on different environments

Operators

- ▶ Control “life cycle” of Kubernetes workloads
 - Operator = CRD(s) + Controller(s)
- ▶ Extend the Kubernetes API with CRDs (Custom Resource Definitions)
- ▶ Simplify to write one (or a few) “custom resource” related YAMLS
- ▶ Acting since day-1 to day-2 operation from deployment to upgrades, through manage
- ▶ ... deployable via an Helm Charts :-)
 - Helm guarantees CRDs are installed before operator
 - Operator configurable via values or ConfigMap

Demo time!



You convinced me!
How to start?

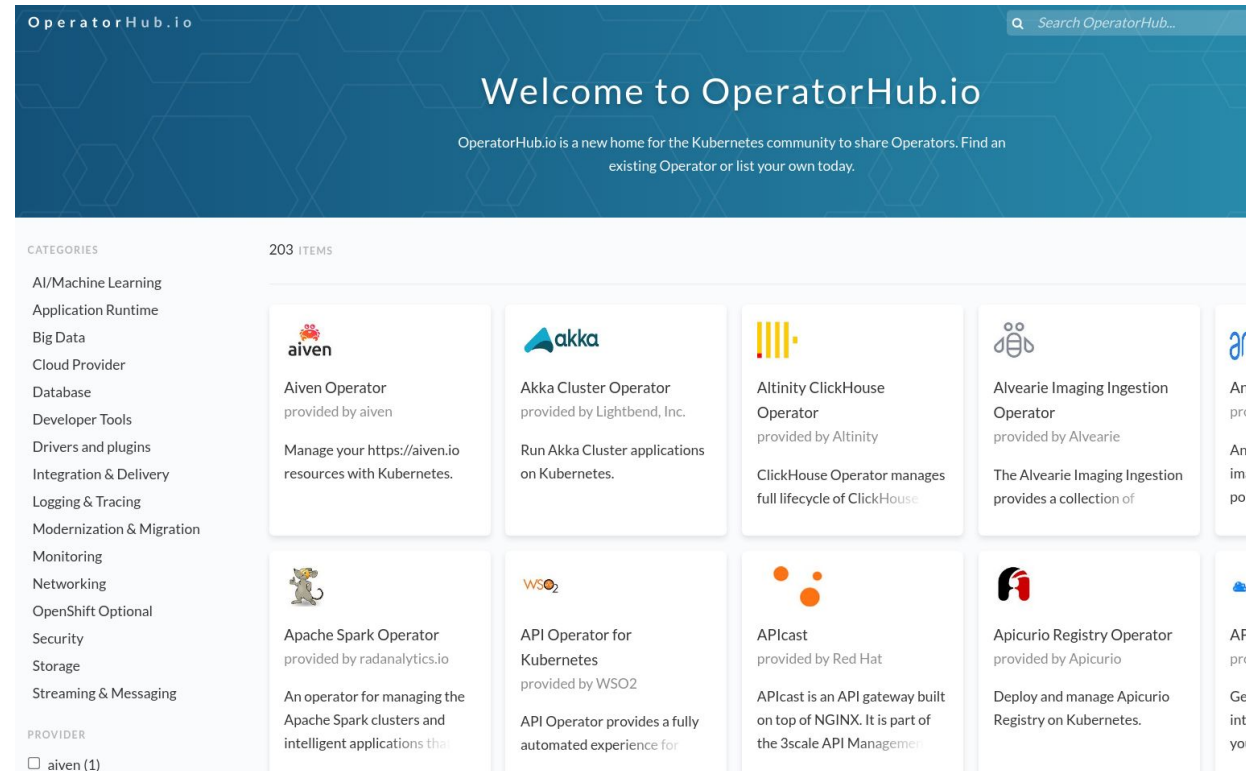
Developing an operator

- ▶ Operator framework
 - Toolkit to manage Kubernetes operator
 - SDK for writing operators in Golang
 - Operator Lifecycle Manager (OLM) to handle ... operators
 - Operator registry to provide operators to OLM
 - ... and much more
- ▶ Java Operator SDK
 - Writing operators in Java
 - Support for Quarkus



OperatorHub.io

- ▶ Home for Kubernetes operators
 - A lot of categories (Database, Streaming & messaging, Logging & Tracing, ...)
- ▶ Installation via Helm Charts or YAML files
- ▶ You can develop your own and provide to the community



Resources

- ▶ [Operator Framework](#)
- ▶ [Java Operator SDK](#)
- ▶ [OperatorHub.io](#)
- ▶ [Strimzi](#)
- ▶ [Apache Kafka](#)
- ▶ [Demo](#)

Thank you!
Questions?