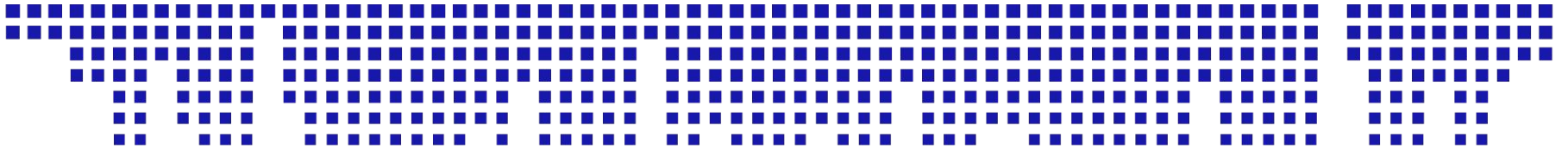
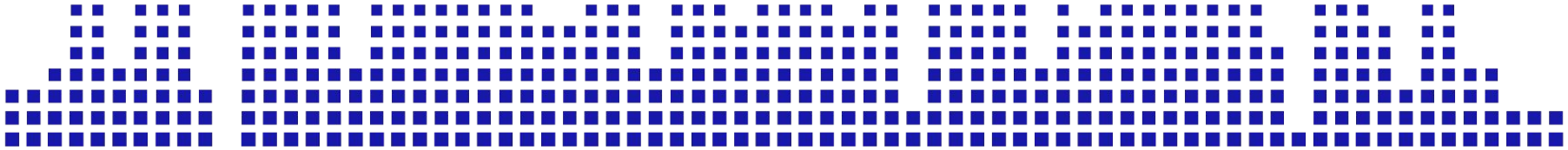
DEVCAST



Kubernetes Operator Pattern:

Workloads automation in the cloud

Paolo Patierno,
Principal Software Engineer @Red Hat



```
apiVersion: v1
kind: PrincipalSoftwareEngineer
metadata:
  name: Paolo Patierno
  namespace: Red Hat, Messaging & Data Streaming
  labels:
    cncf/maintainer: Strimzi
    eclipse/committer: 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"

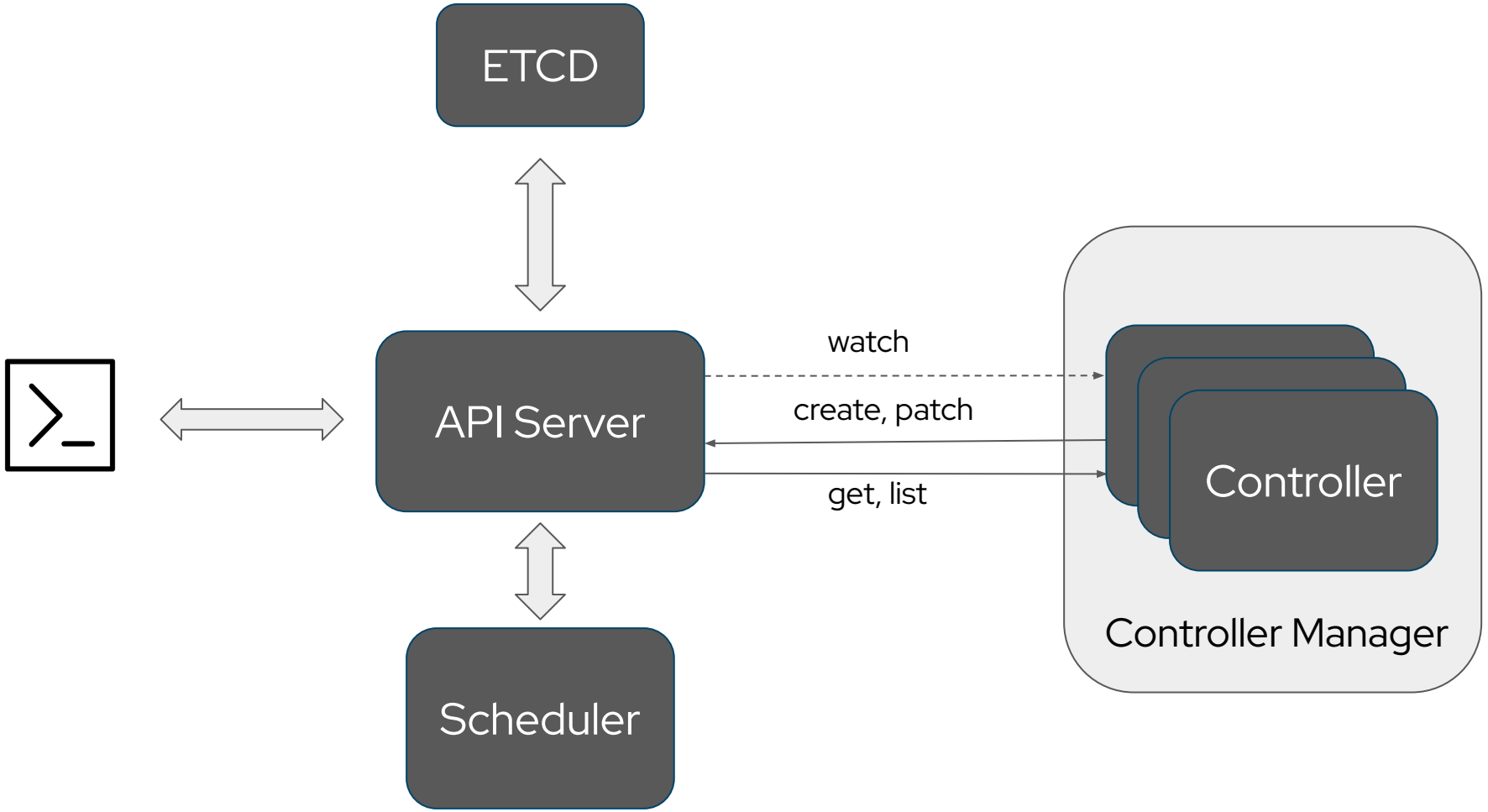
Workloads

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

Workloads

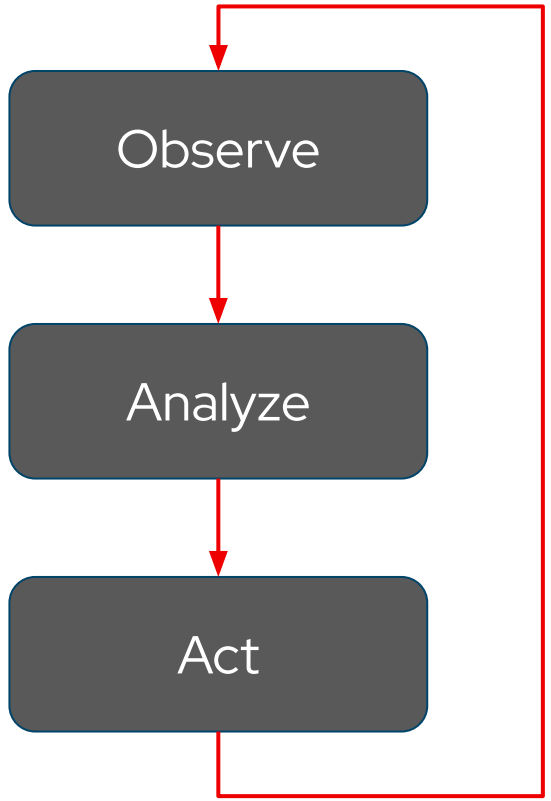
- Don’t use Pod(s) … let’s use something more sophisticated!
- ReplicaSet
  - Guarantees a specific number of running replicas
  - Spins pods, based on a template, if there are not enough
  - Deletes pods if too many match the selector
- Deployment
  - It’s based on ReplicaSet (used to run replicas)
  - 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 ;-)


ETCD
API Server
watch
create, patch
get, list
Controller
Controller Manager
Scheduler

Controller 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



```yaml
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

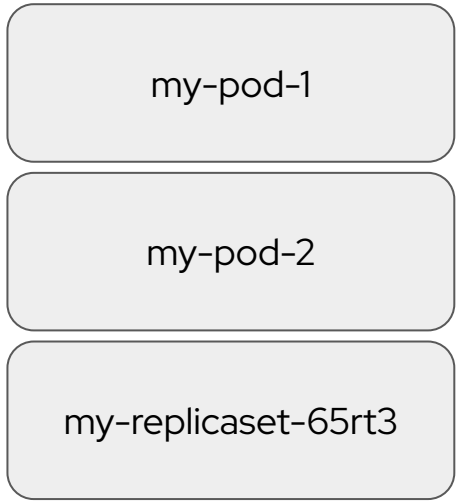
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-1
  labels:
    app: my-app
spec:
  containers:
  - name: my-app
    image: quay.
```

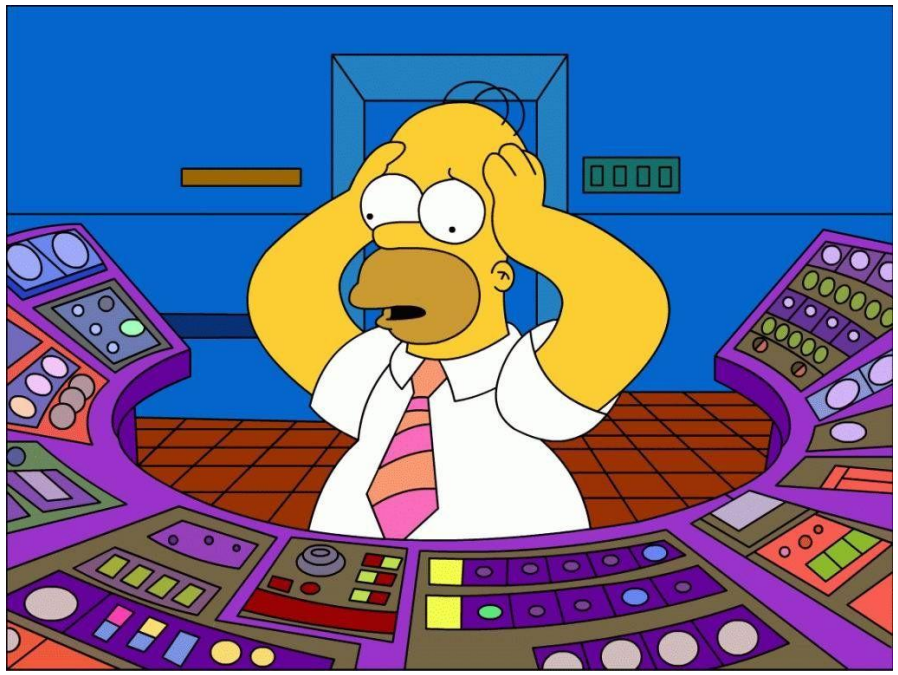
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-2
  labels:
    app: my-app
spec:
  containers:
  - name: my-application
    image: quay.io/ppatierno/my-application:latest
```

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

```yaml
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
```

```yaml
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
```

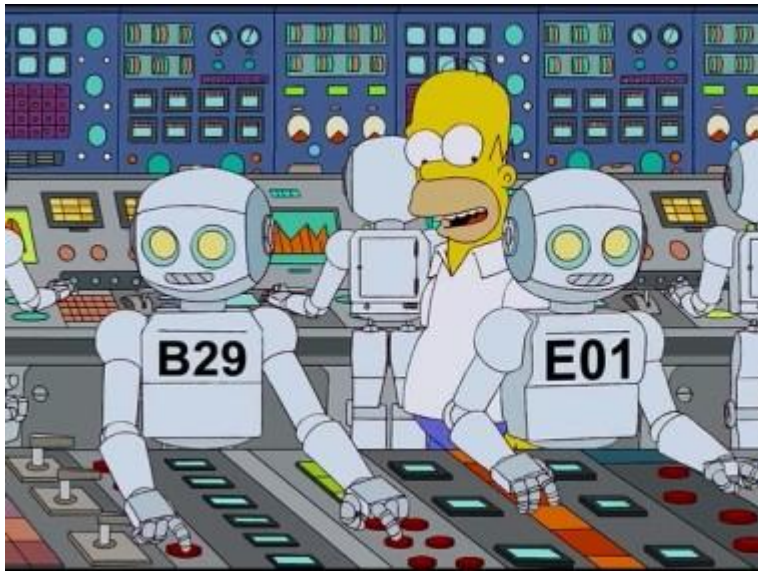


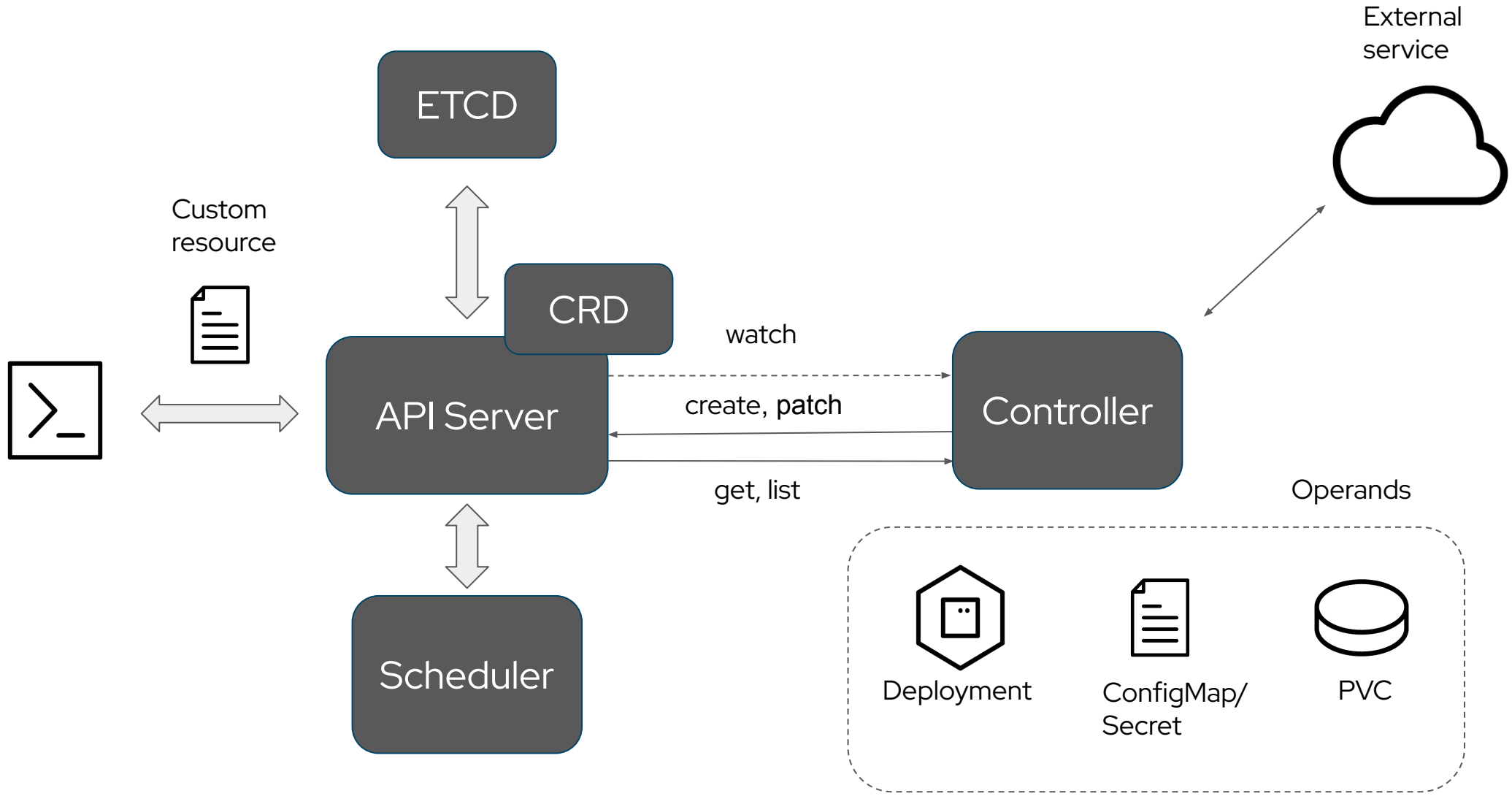


How to automate operating complex applications? The Operator pattern!

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




External
service
ETCD
Custom
resource
CRD
watch
API Server
create, patch
Controller
get, list
Operands
Scheduler
Deployment
ConfigMap/
Secret
PVC

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, …)

THE Apache Kafka operator

STRIMZI

- 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
- http://strimzi.io

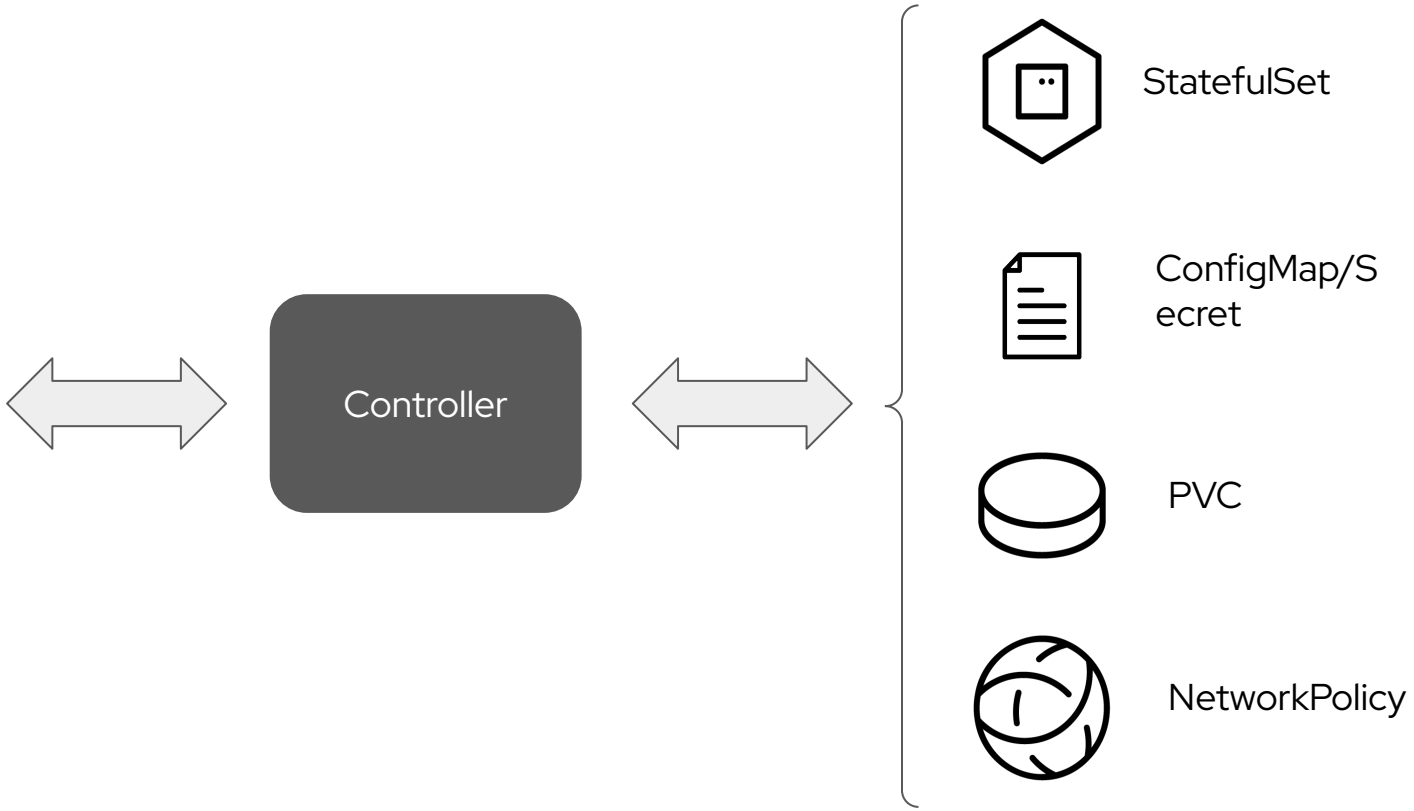
From the Custom Resource Definition ...

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

```yaml
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

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 2.8.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      log.message.format.version: "2.8"
      inter.broker.protocol.version: "2.8"
      ...
    storage:
      type: ephemeral
  zookeeper:
    replicas: 3
    storage:
      type: ephemeral
status:
  ...
  ...
```



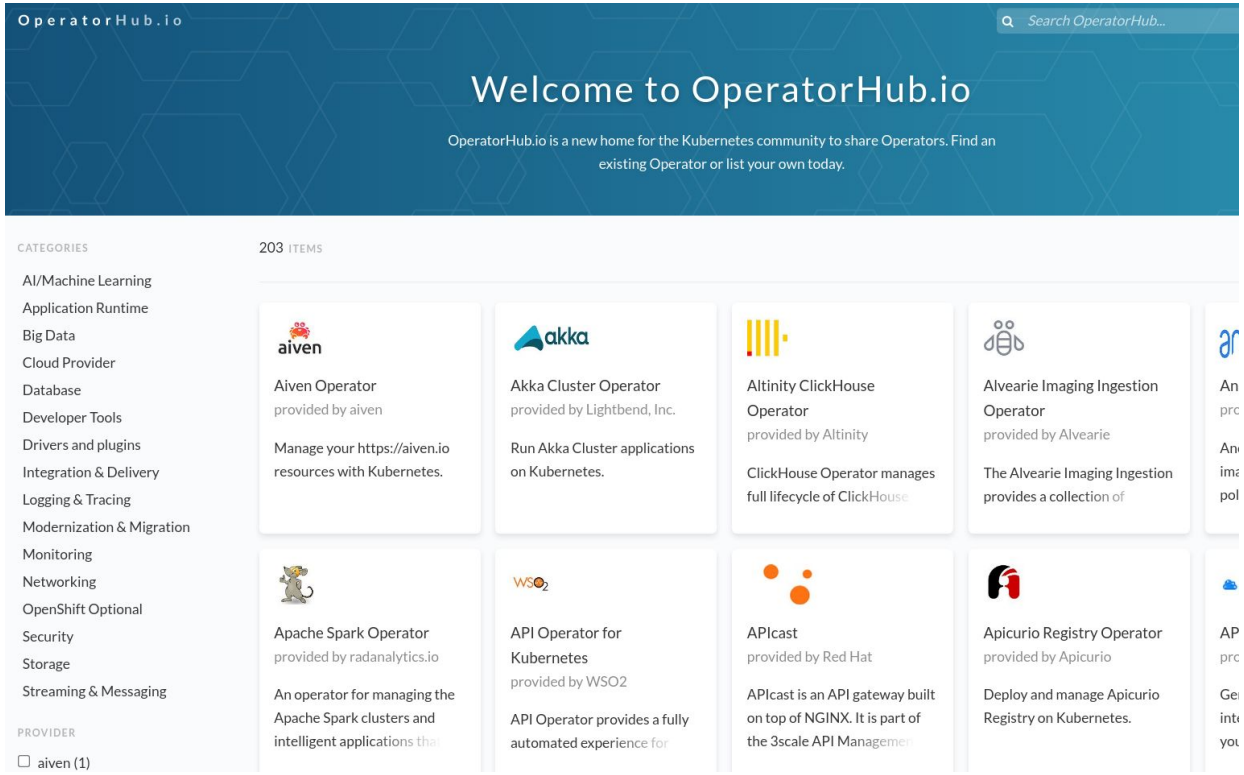
You convinced me!
How to start?

Developing an operator

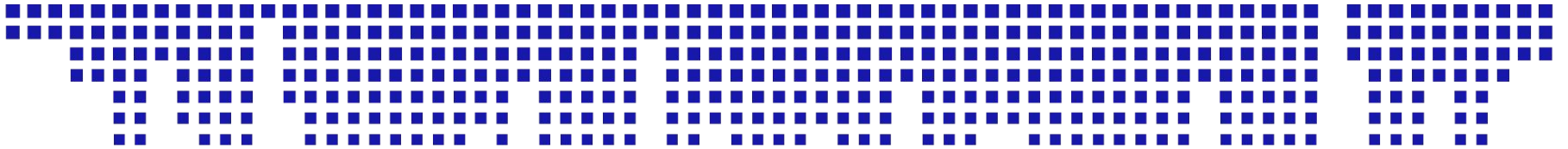
- Operator framework
  - Toolkit to manage Kubernetes operator
  - SDK for writing operators in Golang
  - Operator Lifecycle Manager (OLM) to handle … operators
  - Operator registry to provide operators to OLM
  - … and much more
  - https://sdk.operatorframework.io/
- Java Operator SDK
  - Writing operators in Java
  - Support for Quarkus
  - https://javaoperatorsdk.io/

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
- https://operatorhub.io/



DEVCAST



Thank you!