CLOUD NATIVE
COMPUTING FOUNDATION

PORTO
KCD
GROWING CLOUD NATIVE TOGETHER

Managing Kubernetes workloads: extend the platform with operators

Paolo Patierno,

Senior Principal Software Engineer @ Red Hat

Who am I?

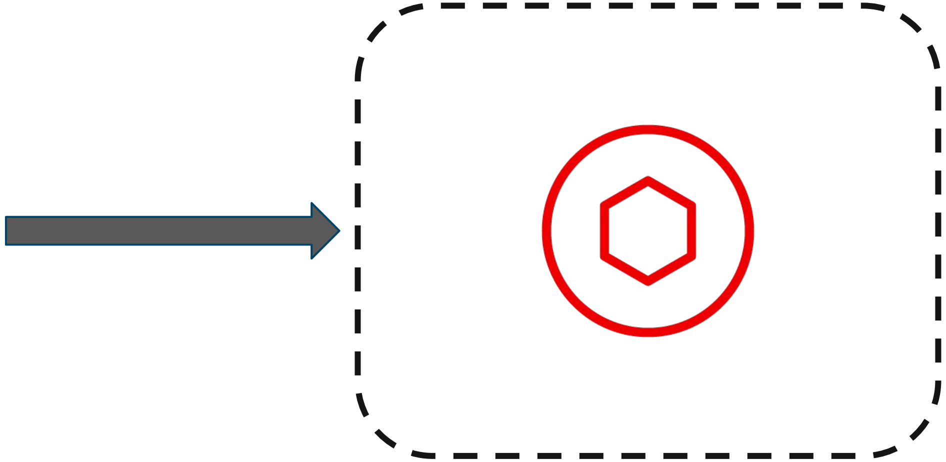
@ppatierno

- Senior Principal Software Engineer @ Red Hat
  - Messaging & data streaming
- CNCF Ambassador
- Strimzi maintainer
- Running, swimming, Formula 1 & MotoGP addicted

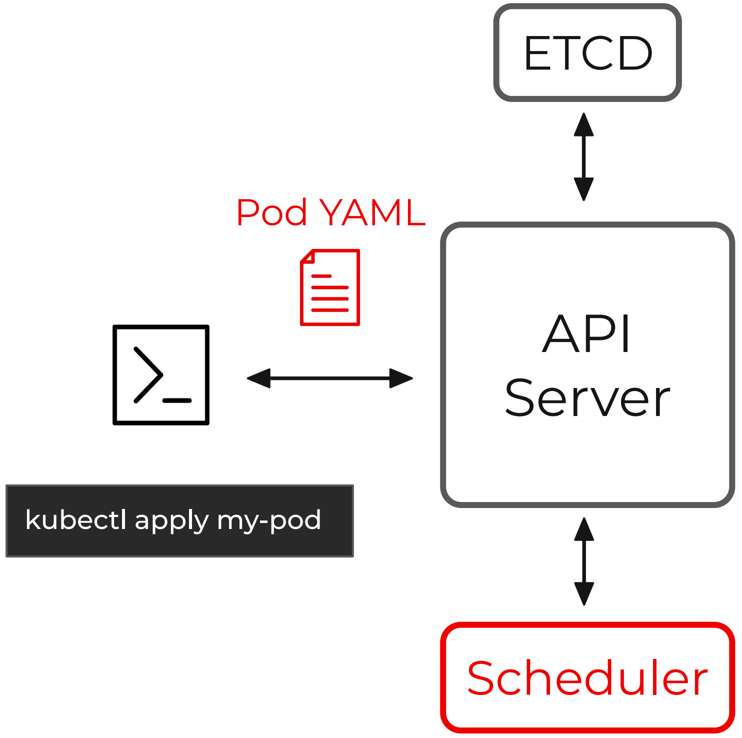
We all know about Kubernetes … right? :-)

Kubernetes is … declarative!

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
spec:
 containers:
 - name: nginx
   image: nginx:1.14.2
   ports:
   - containerPort: 80
```



Kubernetes
cluster



```
kubectl apply my-pod
```

CONTAINERS ... PODS ...
...PODS EVERYWHERE

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

```yaml
apiVersion: v1
kind: Pod
# ...
```

```yaml
apiVersion: apps/v1
kind: ReplicaSet
#
```

```yaml
apiVersion: apps/v1
kind: Deployment
#
```

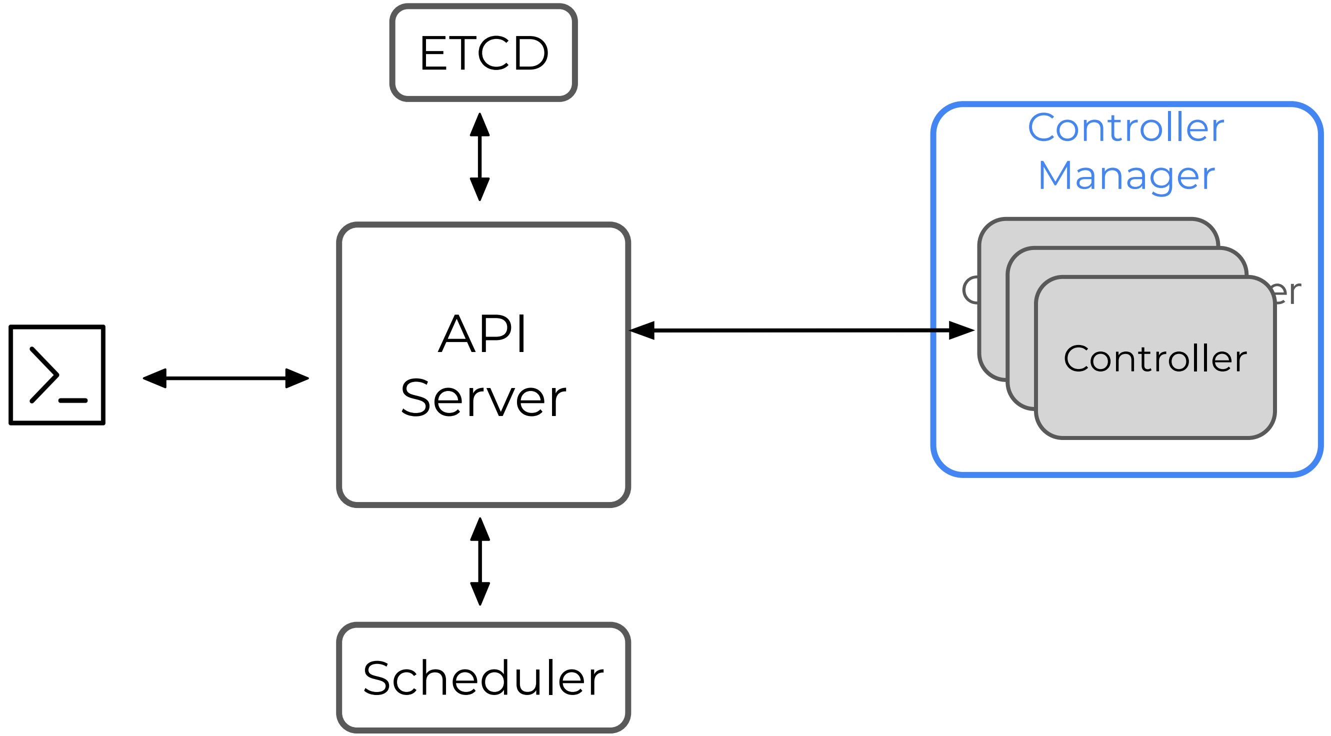
```yaml
apiVersion: apps/v1
kind: StatefulSet
# ...
```

⋱

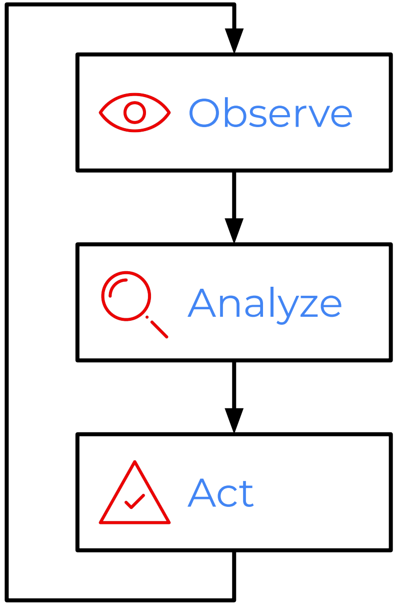
```yaml
apiVersion: batch/v1
kind: Job
# ...
```

How does it work?
Let’s use a controller!!!

…. But not this one ;-)


ETCD
API Server
Scheduler
Controller Manager
Controller

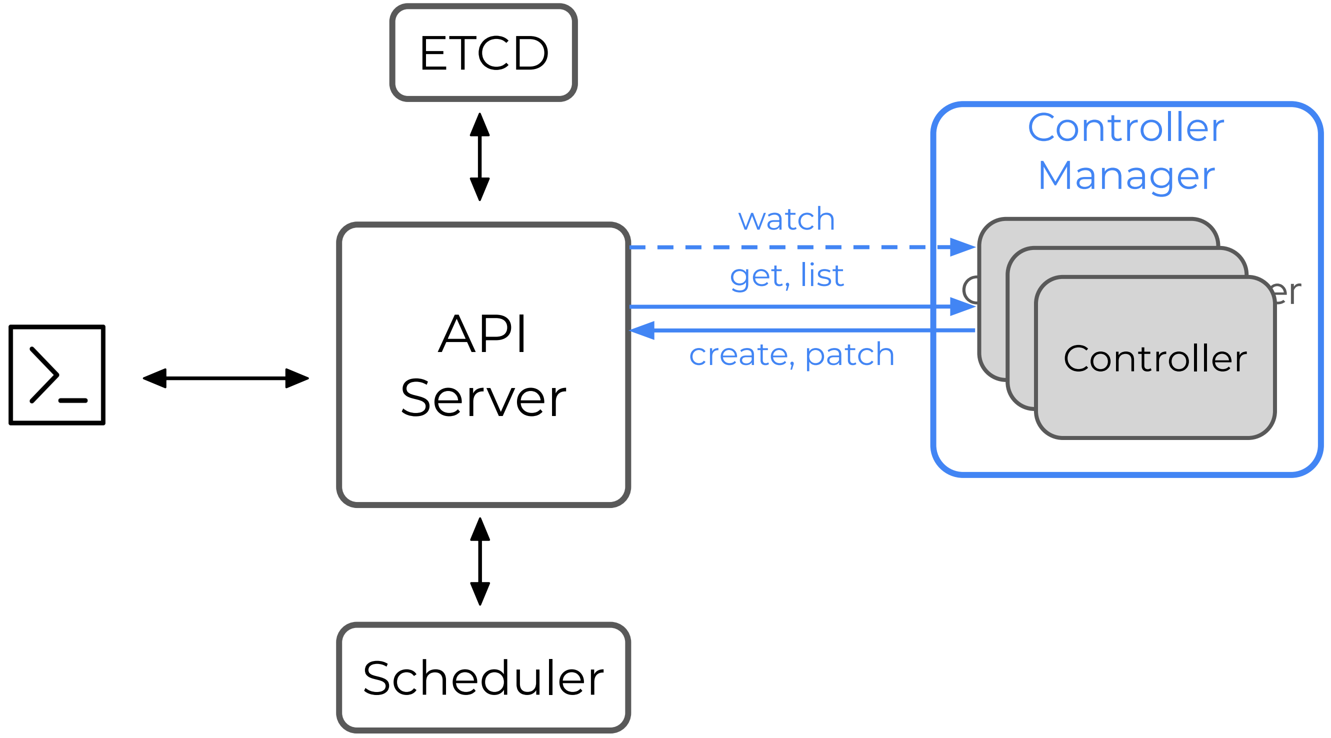
Reconcile Loop



Watch for resource/object creation or changes

Check that the resource/object desired state ("spec") reflects the current state on the cluster

Makes the needed changes


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

Observe

ReplicaSet
resource created

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: my-replicaset
spec:
 replicas: 2
 selector:
   matchLabels:
     app: my-app
 template:
   metadata:
     labels:
       app: my-app
   spec:
     containers:
     - name: my-application
       image: quay.io/devoxxuk/my-application:latest
```

Analyze

2 replicas, spec

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: my-replicaset
spec:
 replicas: 2
 selector:
   matchLabels:
     app: my-app
 template:
   metadata:
     labels:
       app: my-app
   spec:
     containers:
     - name: my-application
       image: quay.io/devoxxuk/my-application:latest
```

Act

Create new pods

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: my-replicaset
spec:
 replicas: 2
 selector:
   matchLabels:
     app: my-app
 template:
   metadata:
     labels:
       app: my-app
   spec:
     containers:
     - name: my-application
       image: quay.io/devoxxuk/my-application:latest
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-bf5zv
 labels:
   app: my-app
spec:
 containers:
 - name: my-application
   image: quay.io/devoxxuk…
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicast-1tf5a
 labels:
   app: my-app
spec:
 # ...
```

What happens if the spec changes?

Observe

ReplicaSet resource updated

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: my-replicaset
spec:
 replicas: 2
 selector:
   matchLabels:
     app: my-app
 template:
   metadata:
     labels:
       app: my-app
   spec:
     containers:
     - name: my-application
       image: quay.io/devoxxuk/my-application:latest
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-bf5zv
 labels:
   app: my-app
spec:
 # ...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-1tf5a
 labels:
   app: my-app
spec:
 # ...
```

Observe

ReplicaSet resource updated

```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/devoxxuk/my-application:latest
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-bf5zv
 labels:
   app: my-app
spec:
 # ...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-1tf5a
 labels:
   app: my-app
spec:
 # ...
```

```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/devoxxuk/my-application:latest
```

Analyze

Search for pods matching resources

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-bf5zv
 labels:
   app: my-app
spec:
 # ...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-1tf5a
 labels:
   app: my-app
spec:
 # ...
```

Act

Create new pod

```
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/devoxxuk/my-application:latest
```

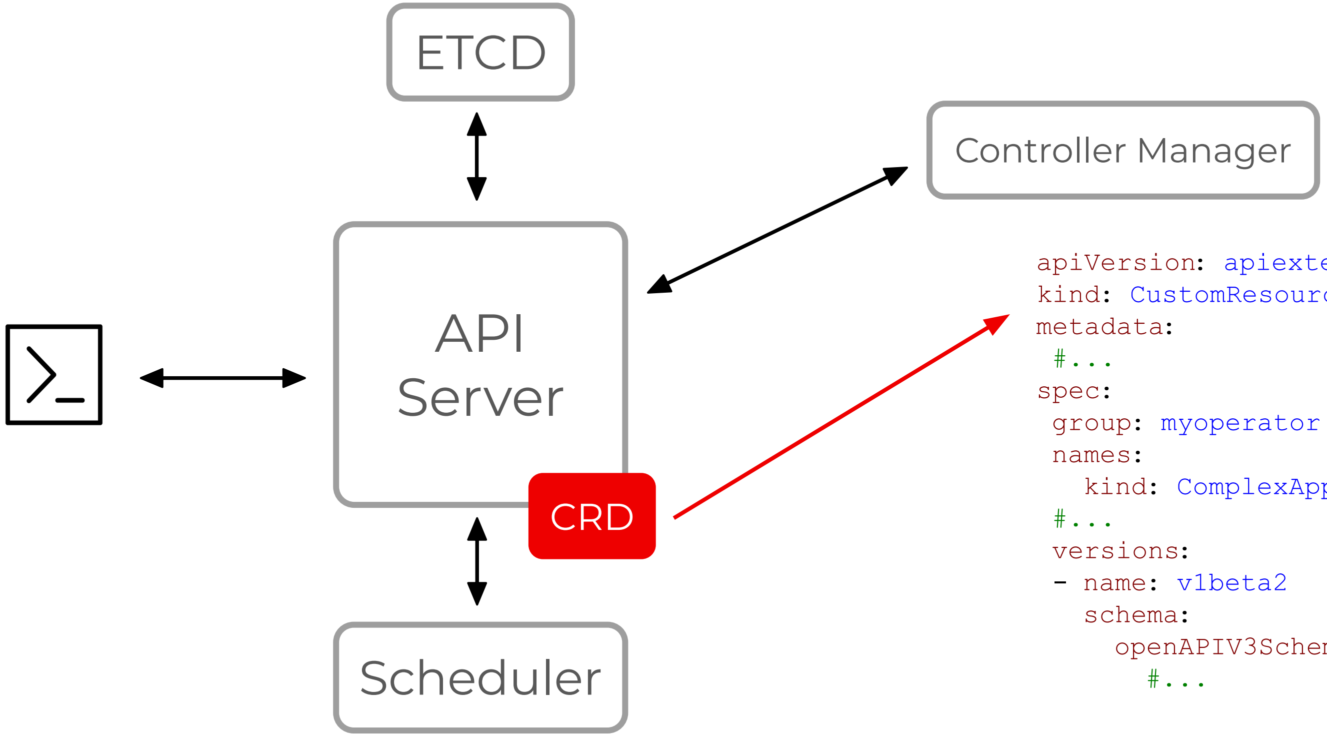
```
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-bf5zv
 labels:
   app: my-app
spec:
 # ...
```

```
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-1tf5a
 labels:
   app: my-app
spec:
 # ...
```

```
apiVersion: v1
kind: Pod
metadata:
 name: my-replicaset-gb65f
 labels:
   app: my-app
spec:
 # ...
```

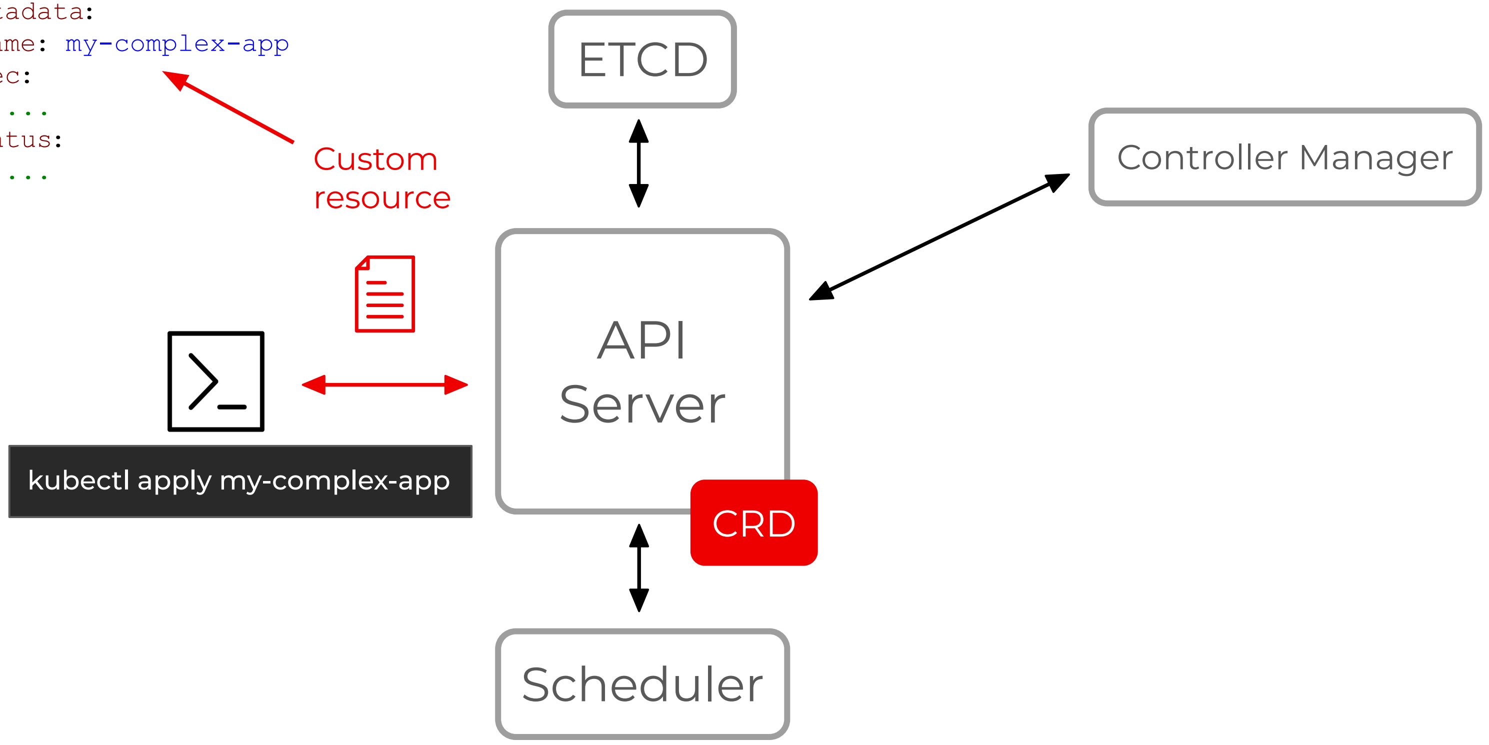
How to automate operating complex “containerized” applications?

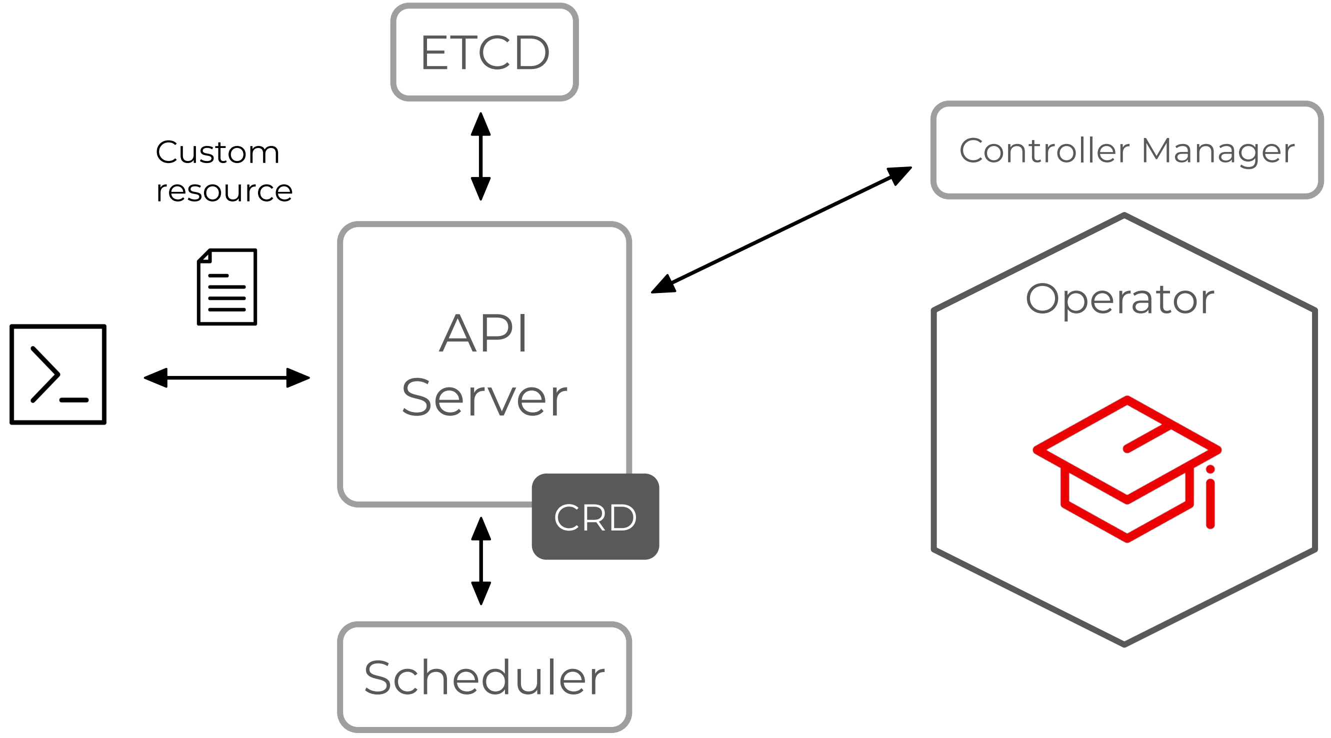
The Operator pattern!

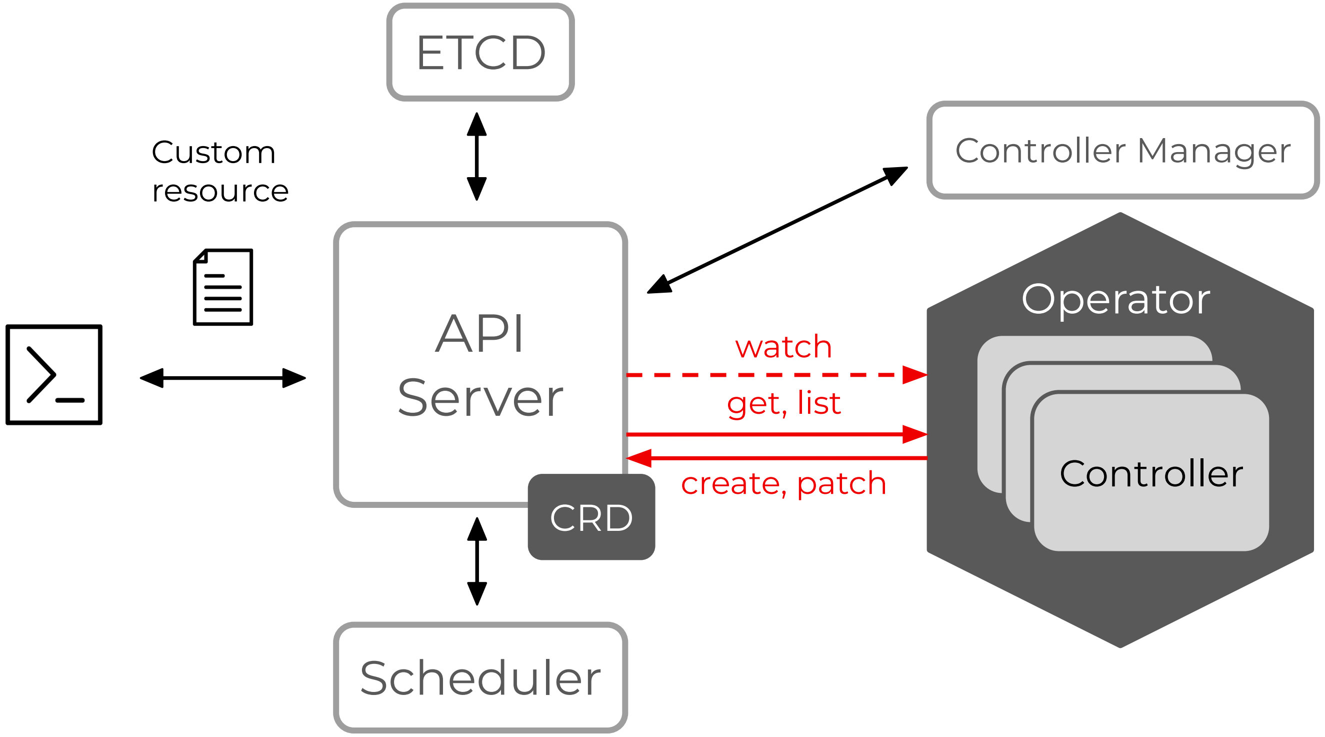


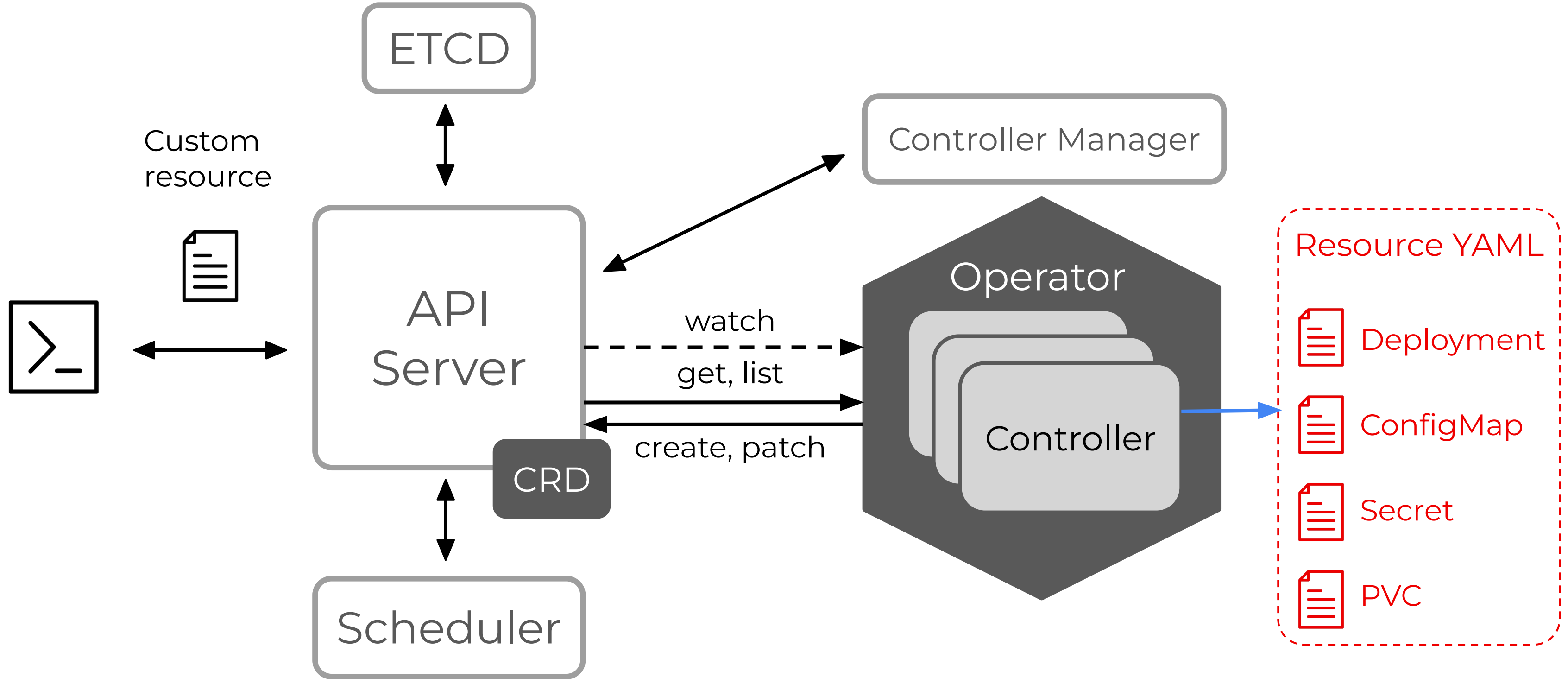
```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
 #...
spec:
 group: myoperator.io
 names:
   kind: ComplexApp
 #...
 versions:
 - name: v1beta2
   schema:
     openAPIV3Schema:
       #...
```

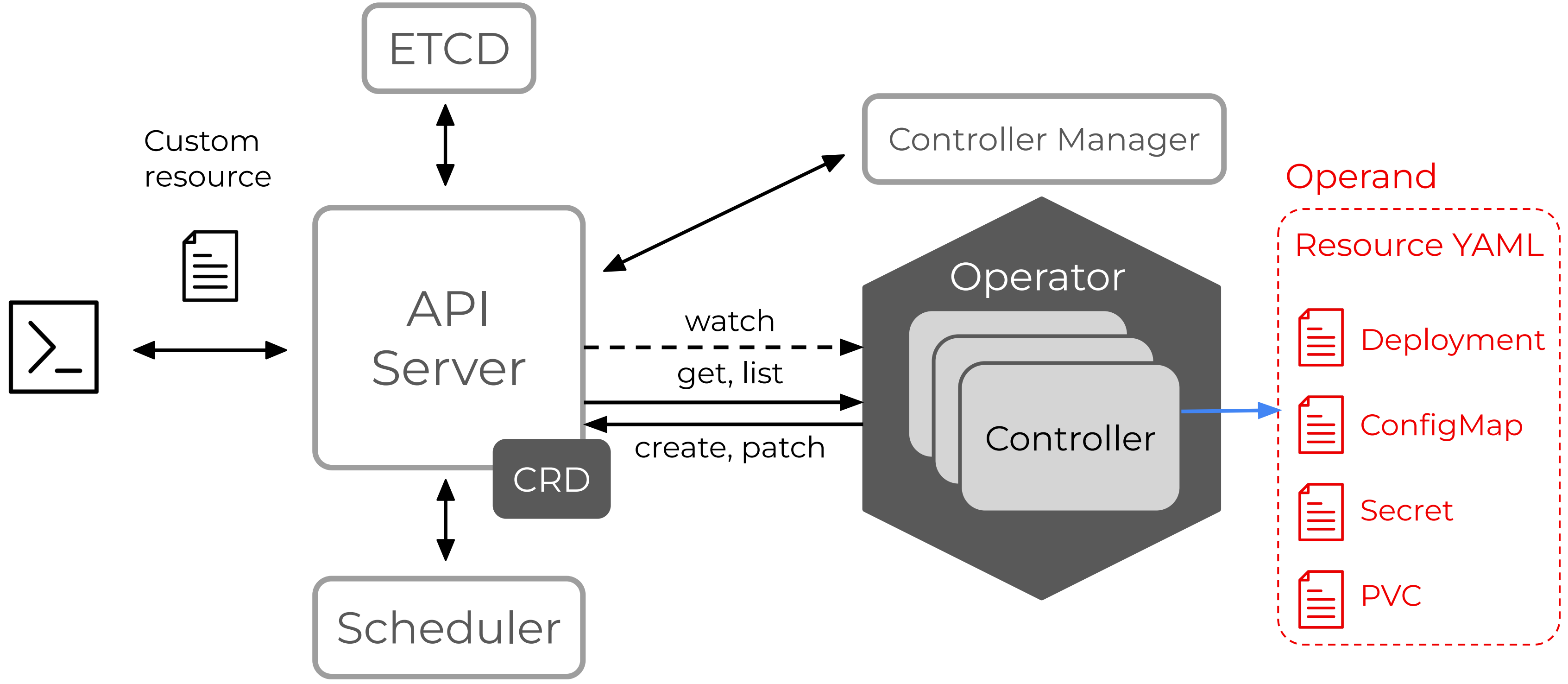
```yaml
apiVersion: myoperator.io/v1
kind: ComplexApp
metadata:
 name: my-complex-app
spec:
 # ...
status:
 # ...
```

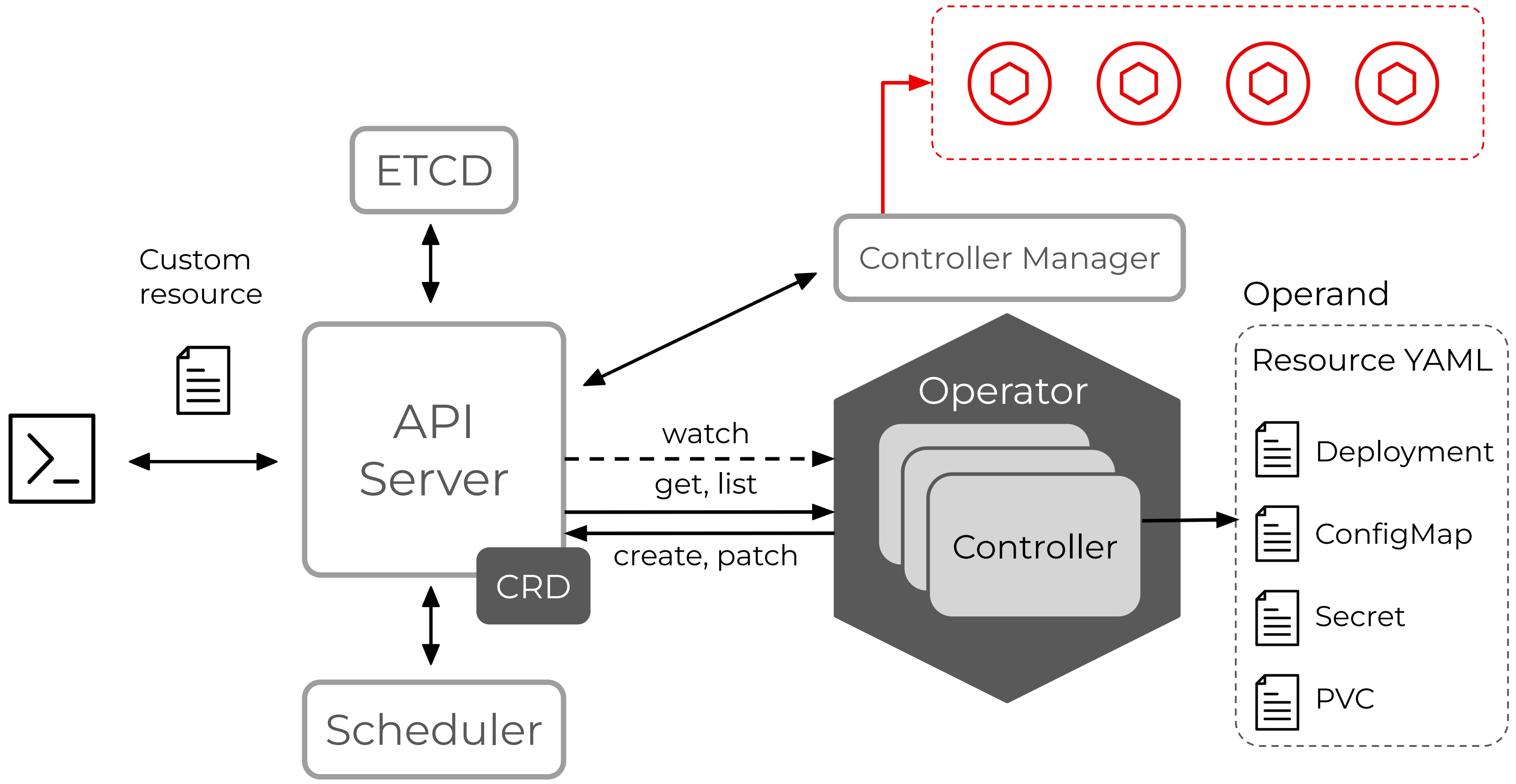


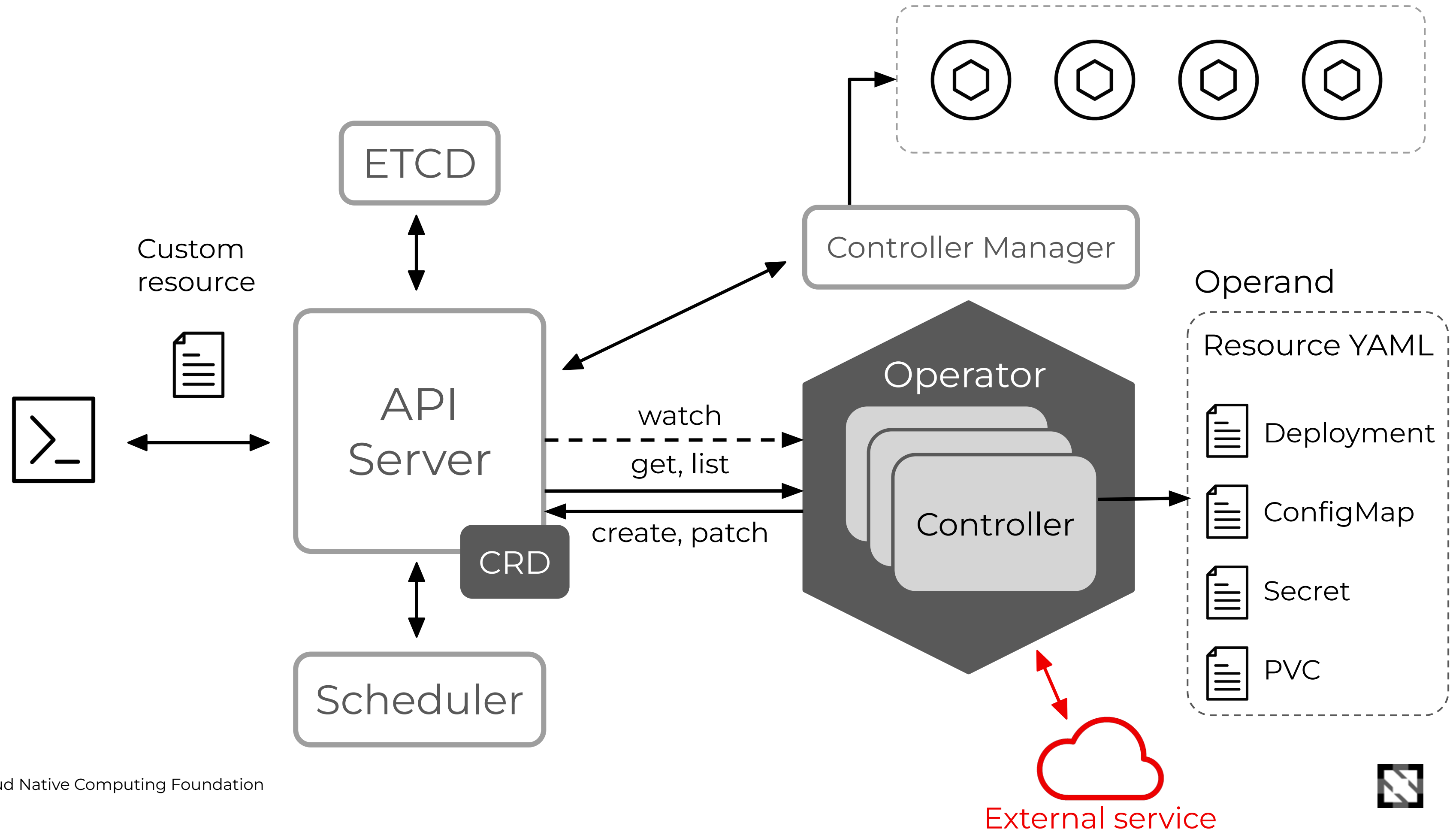

ETCD
Custom resource
Controller Manager
API Server
Operator
CRD
Scheduler


ETCD
Custom resource
Controller Manager
API Server
Operator
watch
get, list
create, patch
Controller
CRD
Scheduler


ETCD
Custom resource
Controller Manager
API Server
CRD
watch
get, list
create, patch
Operator
Controller
Resource YAML
Deployment
ConfigMap
Secret
PVC
Scheduler


ETCD
Custom resource
Controller Manager
Operand
Resource YAML
API Server
Operator
watch
get, list
create, patch
Controller
CRD
Deployment
ConfigMap
Secret
PVC
Scheduler

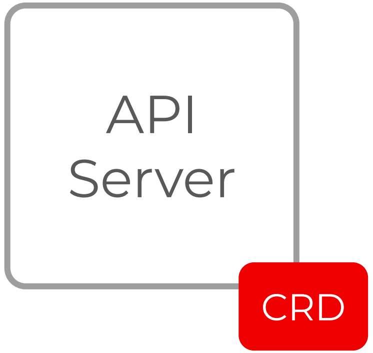

ETCD
Custom resource
Controller Manager
Operand
Resource YAML
API Server
Operator
watch
get, list
Controller
create, patch
CRD
Deployment
ConfigMap
Secret
PVC
Scheduler



ETCD
Custom resource
API Server
CRD
Scheduler
Controller Manager
watch
get, list
create, patch
Operator
Controller
Operand
Resource YAML
Deployment
ConfigMap
Secret
PVC
External service

What if I want to deploy Apache Kafka on Kubernetes?

```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 for the custom resource
           kafka:
             #...
         status:
           # status definition reported back
           # in the custom resource
```



```
GET /apis/kafka.strimzi.io/v1beta2/kafkas/
kubectl get kafka
```

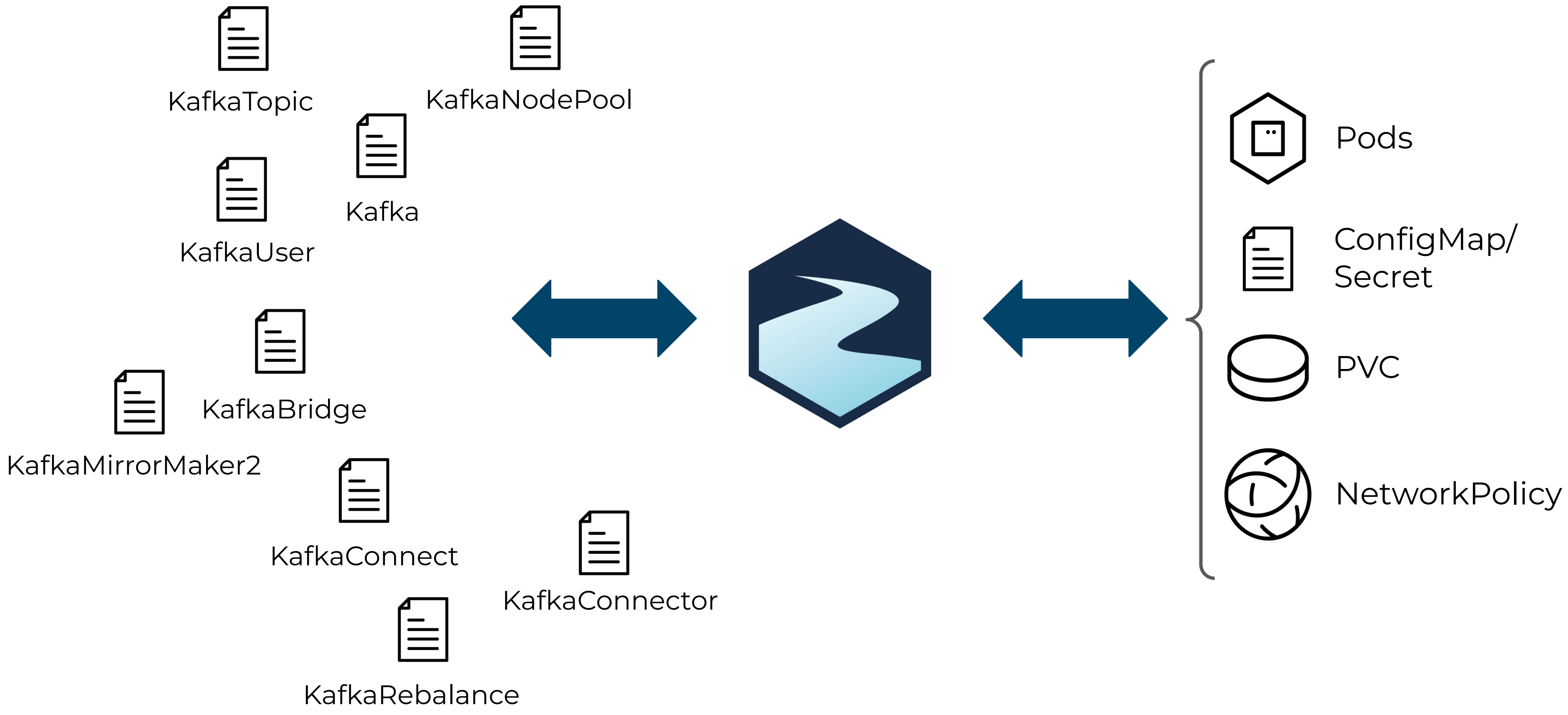
Strimzi

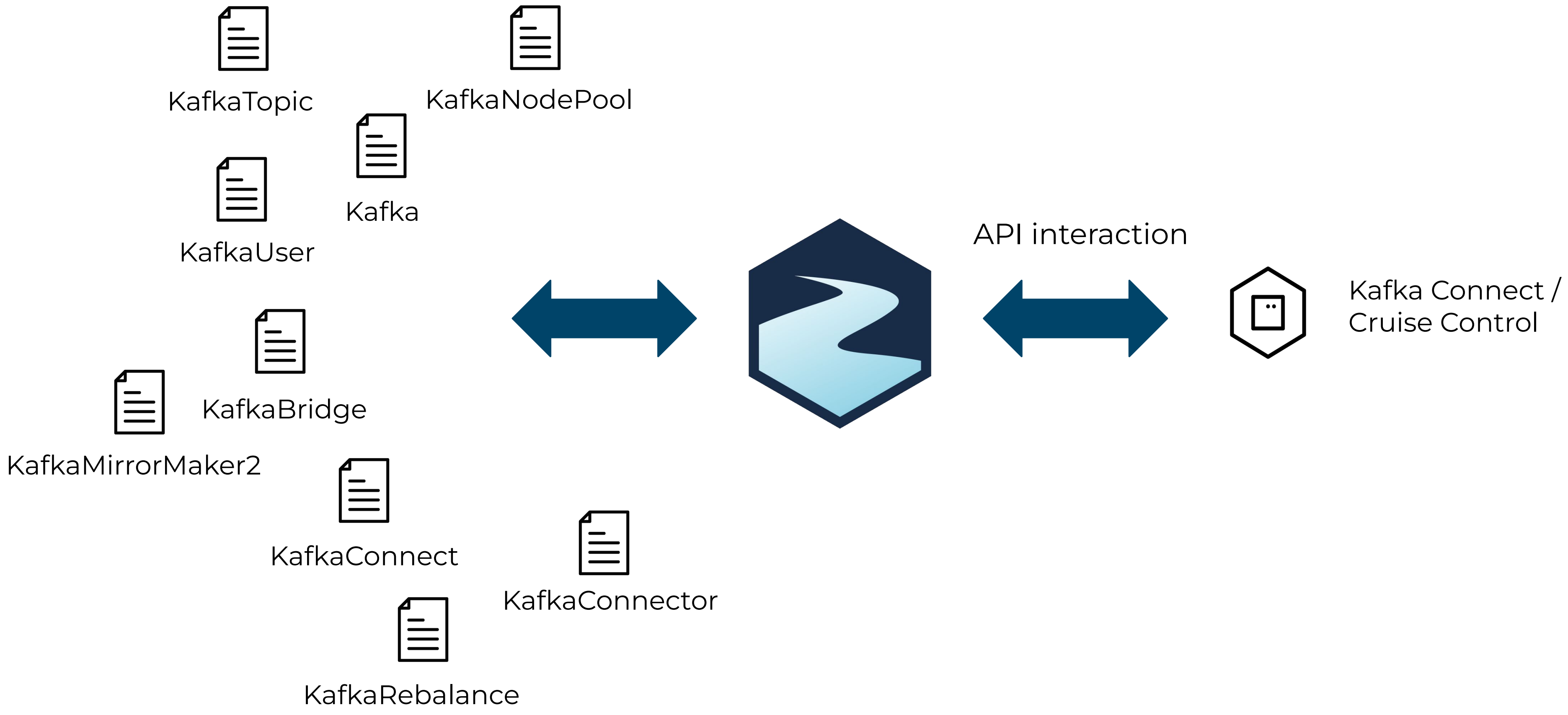
Open Source project (Apache License 2.0)

Focuses on Apache Kafka on Kubernetes

CNCF Incubating Project

strimzi.io


KafkaTopic
KafkaNodePool
Kafka
KafkaUser
KafkaBridge
KafkaMirrorMaker2
KafkaConnect
KafkaConnector
KafkaRebalance
Pods
ConfigMap/
Secret
PVC
NetworkPolicy


KafkaTopic
KafkaNodePool
Kafka
KafkaUser
KafkaBridge
KafkaMirrorMaker2
KafkaConnect
KafkaConnector
KafkaRebalance
API interaction
Kafka Connect /
Cruise Control

Demo time!

Helm

- Relies on Kubernetes built-in resources
- Many YAMLs with customization via templating
- Ideal for day-1 operation (deploying)

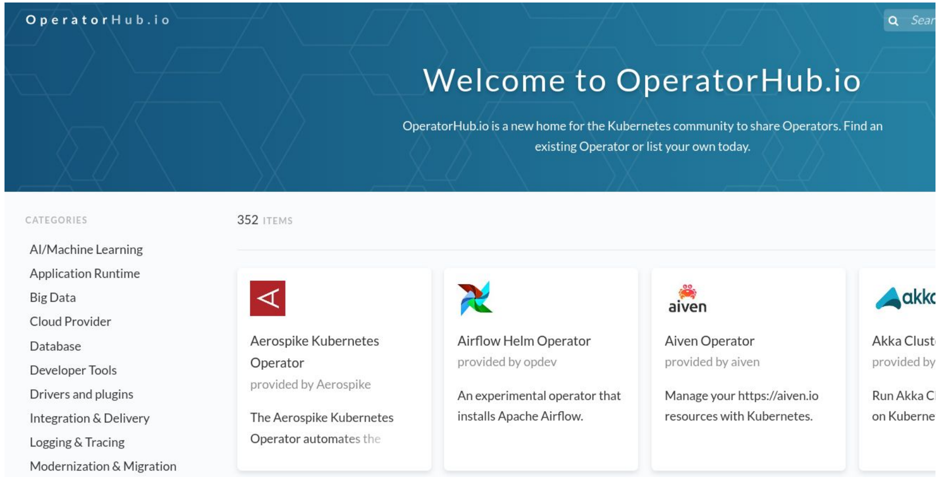
Operator

- Extends the Kubernetes API with CRDs
- One (or a few) “custom resource” YAMLs
- Useful for day-1 and day-2 operations (upgrading, scaling)
- Deployable via Helm charts!

HELM CHARTS
AND MANUAL DEPLOYMENT
STRIMZI
CLUSTER OPERATOR
SERIES 10 / 10
imgflip.com

Where to start?

- Operator SDK ... for writing operators in Go
- Java Operator SKD ... for writing operators in Java
- OperatorHub.io ... provides an operators catalog



Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

CATEGORIES

- AI/Machine Learning
- Application Runtime
- Big Data
- Cloud Provider
- Database
- Developer Tools
- Drivers and plugins
- Integration & Delivery
- Logging & Tracing
- Modernization & Migration

352 ITEMS

Aerospike Kubernetes Operator
provided by Aerospike
The Aerospike Kubernetes Operator automates the

Airflow Helm Operator
provided by opdev
An experimental operator that installs Apache Airflow.

Aiven Operator
provided by aiven
Manage your https://aiven.io resources with Kubernetes.

Akka Clust
provided by
Run Akka C
on Kuberne

OPERATOR FRAMEWORK

[JAVA OPERATOR SDK]

CLOUD NATIVE
COMPUTING FOUNDATION

Thank you!