Operator – SDK 란?

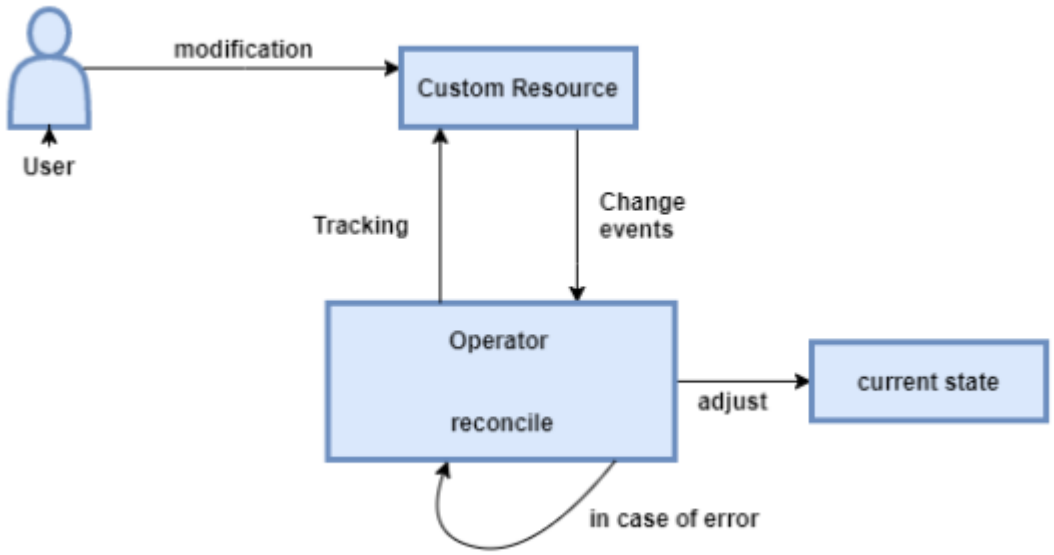
애플리케이션을 패키징, 배포 및 관리하는 것

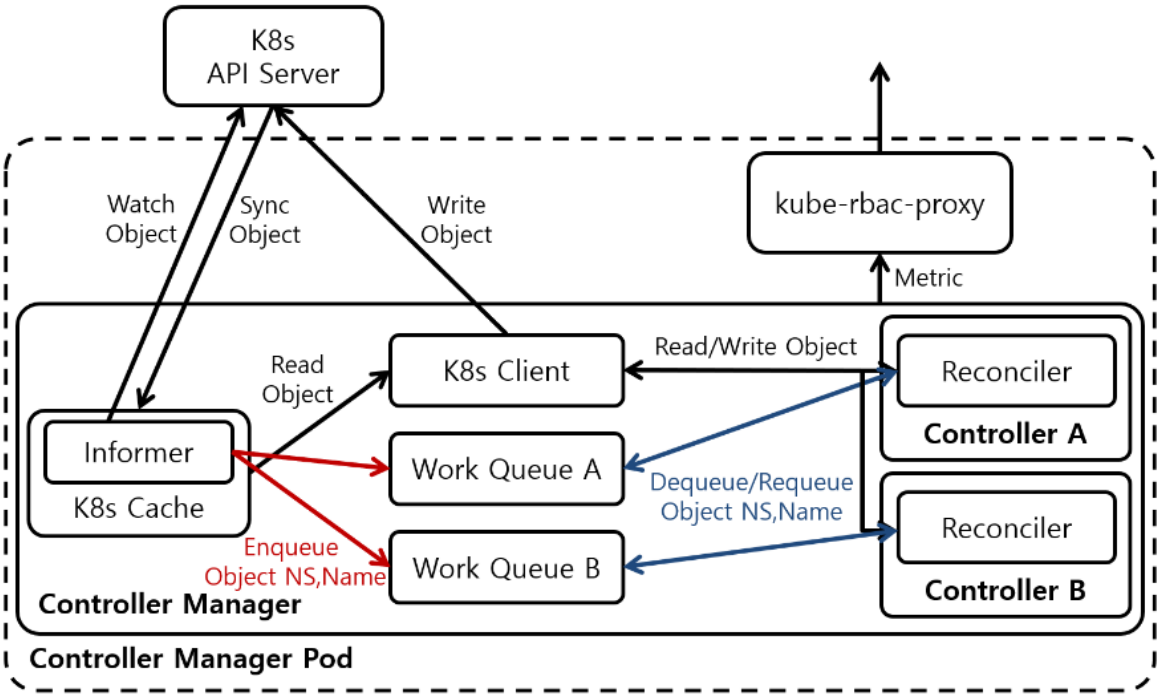
컨트롤러, 커스텀 리소스 생성, 등록 을 자동화 해놓은 것

1. 특정 리소스 : 직접 정의한 **CRD(Custom Resource Definition)** 생성 -> etcd 등록
2. 감지 : CRD의 상태 변화를 감지, CRD desired state 를 맞추는 current state action -> 생성 로직 돌리는 함수

--→ k8s operator 패턴

3. 로직 수행 : object가 원하는 상태가 되도록 로직을 수행, 컨트롤러 (Reconcile 함수가 수행)





[그림 1] Controller Manager Architecture

Operator의 manager 컨테이너 안에는 k8s cache, k8s client, work queue가 있다.

1. 리소스 감지

   1) K8s api server의 부하를 줄이기 위해 **k8s cache** 에서 캐싱한다.

   2) **Imformer**는 operator가 관리하는 리소스를 watch하며, 리소스의 상태변화가 있을 때마다 event로 받는다. (name, namespace)

   3) 이 정보를 **work queue**에 넣는다. Work queue는 컨트롤러 마다 존재한다.

   4) K8s client 는 controller가 k8s api 서버와 통신하기 위한 역할을 수행하며, manager에 하나의 client가 존재한다.

2. 로직 수행

   1) Reconciler는 work queue에 저장된 event를 가져온다.

   2) K8s client를 이용하여 object가 원하는 상태가 되도록 일치시키기 위한 로직을 수행한다.

   3) 실패했을 경우 성공할 때 까지 work queue에 넣고 빼내어 로직을 계속해서 수행한다.

- 예시

```
func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error) {
  // Lookup the Memcached instance for this reconcile request
  memcached := &cachev1alpha1.Memcached{}
  err := r.Get(ctx, req.NamespacedName, memcached)
  ...
}
```

Reconciler는 object의 name, namespace에 대한 정보만 있다.

실제 해당 object를 가져오기 위해서는 k8s client를 통해 가져와야 한다.

Reconcile 함수의 return 값을 보면 ctrl.Result가 있는데, work queue에 다시 넣는 것과 관련된 부분이다.

로직을 수행한 경우에는 빈 큐를 리턴한다.

Ctrl.Result{}

로직을 수행하지 못한 경우 다시 work queue에 집어넣는다.

Ctrl.Result{Requeue:true}

Error 가 있는 경우 error 타입을 반환하고, 아니면 nil 을 반환한다.

로직을 수행한 리턴 값

```
return ctrl.Result{}, nil
```