Introduction to operator pattern

莊家雋

Outline

- Operator Pattern
- 流程
- 二種實作方法
  - Informer
  - Operator-sdk framework
- operator-sdk flow
- Idempotency
- Exercise

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.

Motivation

The Operator pattern aims to capture the key aim of a human operator who is managing a service or set of services. Human operators who look after specific applications and services have deep knowledge of how the system ought to behave, how to deploy it, and how to react if there are problems.

People who run workloads on Kubernetes often like to use automation to take care of repeatable tasks. The Operator pattern captures how you can write code to automate a task beyond what Kubernetes itself provides.

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

使用informer

- 公司有多個部門，每個部門有對應的Namespace

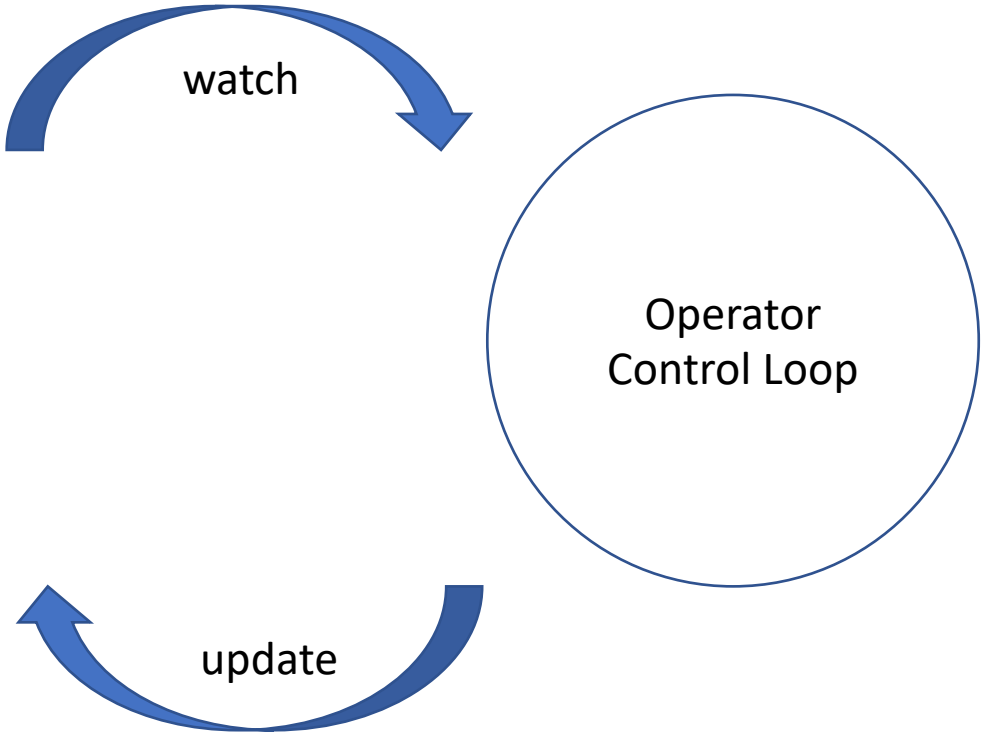
- 建立Namespace時，要手動建立
  - 一個ConfigMap包含共用設定檔
  - 一個Deployment運行部門網站

- 部門裁徹時，要刪除這些建立的資源

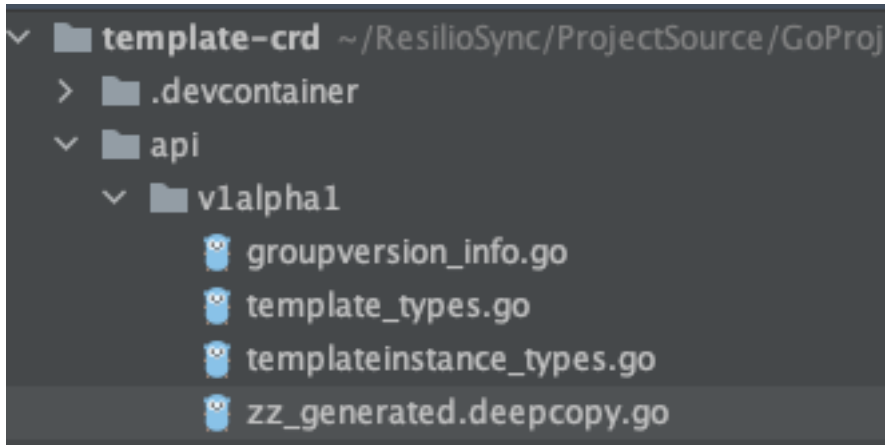
使用Operator

Project CRD

```
apiVersion: xyz.com/v1
kind: Project
metadata:
   name: abc-department
spec:
   image: “abc-web-server:0.1.2”
   password: “P@ssw0rd”
status:
   Ready: True
```



流程



- 設計CRD
- 產生api code (類似k8s.io/api)
  - xxx_type.go
  - zz_generated.deepcopy.go (通常由工具自動產生)
- 開發相對應的operator處理CRD
- operator打包成Image，運行在Cluster內
- 建立CRD Resource

二種實作方法

- 利用Informer
  - 手動設計CRD yaml、重頭開始寫Controller code
  - https://github.com/kubernetes/sample-controller
  - https://tw.wbsnail.com/p/dive-into-kubernetes-informer-crd-informer

- 利用現成的operator-sdk framework工具
  - 自動產生Controller程式碼樣版、CRD yaml樣版
  - https://sdk.operatorframework.io/docs/building-operators/golang/tutorial/
  - https://lailin.xyz/post/operator-03-kubebuilder-tutorial.html

二種實作方法

- 比較
  - https://itnext.io/under-the-hood-of-the-operator-sdk-eebc8fdeebbf

Differences:

1. Compared to the start from scratch approach, the directory structure generated by Operator SDK is significantly different.
2. In the start from scratch approach, typed Client and Lister for our CRDs are generated by update-codegen script. Operator SDK does not generate these. Instead, it internally creates resource clients by utilizing discovery and REST mapping functions offered by the client-go library.

利用Informer

- 原生Resource的factory
  - 建立原生Deployment的informer
- CRD ‘Foo‘ 的Factory
  - 建立Foo的informer

```
kubeInformerFactory := kubeinformers.NewSharedInformerFactory(kubeClient, time.Second*30)
exampleInformerFactory := informers.NewSharedInformerFactory(exampleClient, time.Second*30)

controller := NewController(kubeClient, exampleClient,
        kubeInformerFactory.Apps().V1().Deployments(),
        exampleInformerFactory.Samplecontroller().V1alpha1().Foos())
```

```go
// NewController returns a new sample controller
func NewController(
        kubeclientset kubernetes.Interface,
        sampleclientset clientset.Interface,
        deploymentInformer appsinformers.DeploymentInformer,
        fooInformer informers.FooInformer) *Controller {

        // Create event broadcaster
        // Add sample-controller types to the default Kubernetes Scheme so Events can be
        // logged for sample-controller types.
        utilruntime.Must(samplescheme.AddToScheme(scheme.Scheme))
        klog.V(4).Info("Creating event broadcaster")
        eventBroadcaster := record.NewBroadcaster()
        eventBroadcaster.StartStructuredLogging(0)
        eventBroadcaster.StartRecordingToSink(&typedcorev1.EventSinkImpl{Interface: kubeclientset.CoreV1().Events("")})
        recorder := eventBroadcaster.NewRecorder(scheme.Scheme, corev1.EventSource{Component: controllerAgentName})

        controller := &Controller{
                kubeclientset:     kubeclientset,
                sampleclientset:   sampleclientset,
                deploymentsLister: deploymentInformer.Lister(),
                deploymentsSynced: deploymentInformer.Informer().HasSynced,
                foosLister:        fooInformer.Lister(),
                foosSynced:        fooInformer.Informer().HasSynced,
                workqueue:         workqueue.NewNamedRateLimitingQueue(workqueue.DefaultControllerRateLimiter(), "Foos"),
                recorder:          recorder,
        }

        klog.Info("Setting up event handlers")
        // Set up an event handler for when Foo resources change
        fooInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
                AddFunc: controller.enqueueFoo,
                UpdateFunc: func(old, new interface{}) {
                        controller.enqueueFoo(new)
                },
        })
```

利用operator-sdk: CRD

```
# 初始化project
$ operator-sdk init --domain example.com --repo github.com/example/memcached-operator
```

```
# 建立resource & controller 樣版
$ operator-sdk create api --group cache --version v1alpha1 --kind Memcached --resource --controller
```

```
# 修改 CRD的structure，依需求加上kubebuilder的marker
$ vim api/v1alpha1/memcached_types.go
```

```
# 有修改memcached_types.go，都要執行

# 產生 zz_generated.deepcopy.go
$ make generate

# 產生 CRD的manifest
$ make manifest
```

利用operator-sdk: Controller

```
# 修改 Controller的程式邏輯。 實作 Reconcile(), 依需求加上kubebuilder的marker
$ vim controllers/memcached_controller.go
```

```
# 若Controller裡有修改kubebuilder的marker，要重新產生manifest

# 產生 Controller 的 manifest
$ make manifest
```

```go
// Reconcile is part of the main kubernetes reconciliation loop which aims to
// move the current state of the cluster closer to the desired state.
// TODO(user): Modify the Reconcile function to compare the state specified by
// the Template object against the actual cluster state, and then
// perform operations to make the cluster state reflect the state specified by
// the user.
//
// For more details, check Reconcile and its Result here:
// - https://pkg.go.dev/sigs.k8s.io/controller-runtime@v0.11.0/pkg/reconcile
func (r *TemplateReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {...}
```

OwnerReference

- 指定Owner 和 Dependent的關係
- 刪除Owner時， Dependent同時被刪除

```
ctrl.SetControllerReference(s, cm, r.Scheme)
```

```
oc get pod coredns-6d4b75cb6d-m9jjr -o yaml | yq e 'del(.metadata.managedFields)' -
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2022-05-12T05:14:38Z"
  generateName: coredns-6d4b75cb6d-
  labels:
    k8s-app: kube-dns
    pod-template-hash: 6d4b75cb6d
  name: coredns-6d4b75cb6d-m9jjr
  namespace: kube-system
  ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: true
      controller: true
      kind: ReplicaSet
      name: coredns-6d4b75cb6d
      uid: 0061f051-b359-43d7-91c5-046535aa329d
  resourceVersion: "971877"
  uid: f29b8cb5-7baf-48cc-bd24-6805fb6df72f
```

```
oc get rs coredns-6d4b75cb6d -o yaml | yq e 'del(.metadata.managedFields)' -
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  annotations:
    deployment.kubernetes.io/desired-replicas: "2"
    deployment.kubernetes.io/max-replicas: "3"
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2022-05-12T05:14:38Z"
  generation: 1
  labels:
    k8s-app: kube-dns
    pod-template-hash: 6d4b75cb6d
  name: coredns-6d4b75cb6d
  namespace: kube-system
  ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: true
      controller: true
      kind: Deployment
      name: coredns
      uid: 161f6a46-0992-4b99-b3b2-d8be75d7ea52
  resourceVersion: "971881"
  uid: 0061f051-b359-43d7-91c5-046535aa329d
```

```
oc get deployments.apps coredns -o yaml | yq e 'del(.metadata.managedFields)' -
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2022-05-12T05:14:23Z"
  generation: 1
  labels:
    k8s-app: kube-dns
  name: coredns
  namespace: kube-system
  resourceVersion: "971882"
  uid: 161f6a46-0992-4b99-b3b2-d8be75d7ea52
```

利用operator-sdk: run-out-cluster

```
# 安裝CRD, (要先執行make generate manifest)
$ make install
```

```
# 本地運行Controller
$ make run
```

```
#依自定的CRD要求， 編輯範例CRD
$ vim config/samples/cache_v1alpha1_Memcached.yaml

$ kubectl apply –f config/samples/cache_v1alpha1_Memcached.yaml
```

operator-sdk產生的Makefile裡，會依kubebuild marker, 利用make manifest 產生所需要的yaml 檔 (包含RBAC相關的yaml) , 所以不需手動撰寫YAML。

利用operator-sdk: run-in-cluster

```
# 修改預設的Image名

$ vim Makefile

-IMG ?= controller:latest
+IMG ?= $(IMAGE_TAG_BASE):$(VERSION)
```

```
# 編譯 & 建立Image & 上傳到 image registry
$ make docker-build docker-push
```

```
# 部署
$ make deploy
```

```
# 編譯 & 建立Image
$ make docker-build

# 上傳到 kind
$ kind load docker-image controller:latest

# 部署
$ make deploy
```

8. 執行 `make deploy` 或 `make all -f .validate/Makefile` 前，請注意
   - operator-sdk 產生的Makefile，無需做任何變更
   - 利用 `kind load docker-image controller:latest` 將建立出來的image載入至kind
   - `config/manager/manager.yaml` 中，`image: controller:latest` 下面請多加一行 `imagePullPolicy: IfNotPresent`

```
...
 image: controller:latest
 imagePullPolicy: IfNotPresent
...
```

   - `config/rbac/kustomization.yaml` 裡的倒數四行 `- auth_proxy_*.yaml` 註解掉

流程

```yaml
apiVersion: sample.ntcu.edu.tw/v1alpha1
kind: Sample
metadata:
  name: sample-sample
spec:
  image: alpine
  key: foo
  value: bar-1
```

- 設計CRD
- 產生api code (類似k8s.io/api)
- 開發相對應的operator處理CRD
- operator打包成Image，運行在Cluster內
- 建立CRD Resource

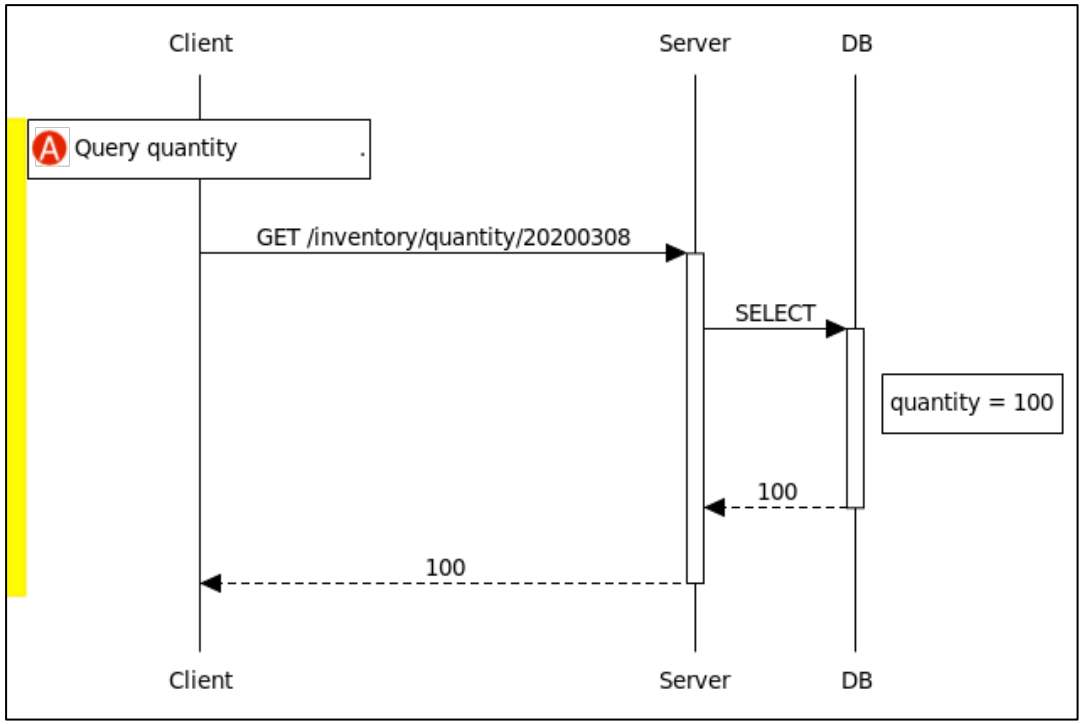
- 修改 sample_types.go
- make generate
- Controller 的 Reconcile()
- make docker-build & make deploy

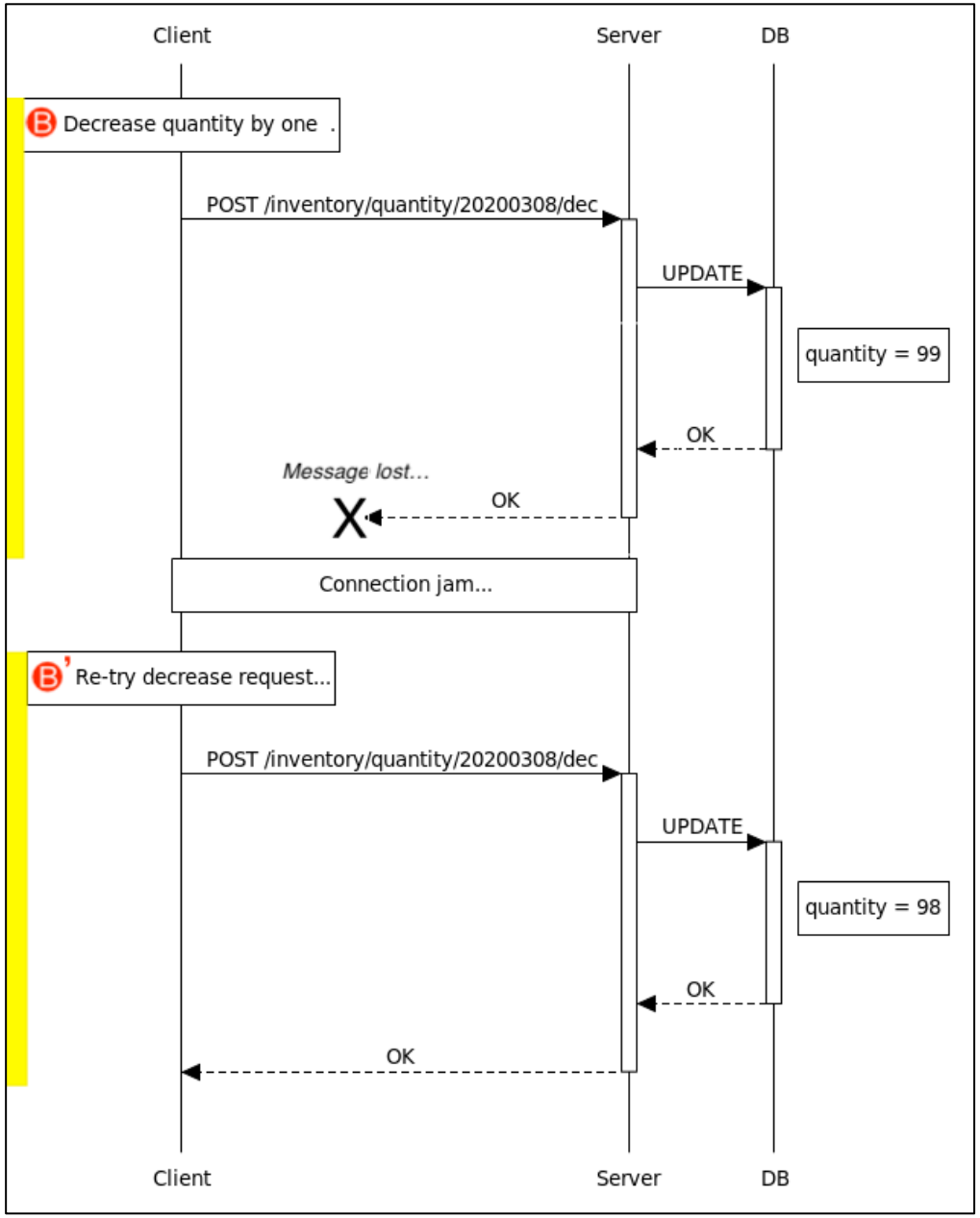
- kubectl apply -f config/samples/sample_v1alpha1_sample.yaml

Idempotent

Develop idempotent reconciliation solutions

When developing operators, it is essential for the controller's reconciliation loop to be idempotent. By following the Operator pattern you will create Controllers which provide a reconcile function responsible for synchronizing resources until the desired state is reached on the cluster. Breaking this recommendation goes against the design principles of controller–runtime and may lead to unforeseen consequences such as resources becoming stuck and requiring manual intervention.





https://william-yeh.net/post/2020/03/idempotency-key-test/

https://sdk.operatorframework.io/docs/best-practices/common-recommendation/

Case Study: Cert manager

- Let’s Encrypt HTTP-01 Challenge

```yaml
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: acme-crt
spec:
  secretName: acme-crt-secret
  dnsNames:
  - foo.example.com
  - bar.example.com
  issuerRef:
    name: letsencrypt-prod
    # We can reference ClusterIssuers by changing the kind here.
    # The default value is Issuer (i.e. a locally namespaced Issuer)
    kind: Issuer
    group: cert-manager.io
```

5. 第二個挑戰：`http-01`

這個挑戰要你在官網建立一個**特殊網址路徑**的**文字檔案**，而且必須可以讓 Let's Encrypt 網站能夠公開存取該網址，而且一定只能走 **Port 80** 進行 HTTP 連線，不能使用任何其他埠號，如此一來才能驗證你就是該網站的擁有者！

網址路徑：`/.well-known/acme-challenge/IKibDaF4-FHZoGw1U6JTyGlBDMOtE-cQCFw13e4FaUc`

檔案內容：

`IKibDaF4-FHZoGw1U6JTyGlBDMOtE-cQCFw13e4FaUc.plEmWe4UXqKWJvuRWXDnZDtkeEh2omjTeQWuZ`

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Create a file containing just this data:

IKibDaF4-FHZoGw1U6JTyGlBDMOtE-cQCFw13e4FaUc.plEmWe4UXqKWJvuRWXDnZDtkeEh2omjTeQWuZ
HEKan4

And make it available on your web server at this URL:

http://angular.tw/.well-known/acme-challenge/IKibDaF4-FHZoGw1U6JTyGlBDMOtE-cQCFw1
3e4FaUc

(This must be set up in addition to the previous challenges; do not remove,
replace, or undo the previous challenge tasks yet.)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Press Enter to Continue
```

> 注意：網站一定要能夠接聽 Port 80 的 HTTP 連接喔！

最後用瀏覽器確定 `http-01` 挑戰的網址可以順利打開，才能按下 `Enter` 繼續！

> 照理說，這個步驟其實很容易完成，就建立幾個資料夾與一個文字檔案而已。但我的網站不小心在 IIS 設定了一個**虛擬目錄**(Virtual Directory)，導致這個 `.well-known` 目錄無法存取該檔案。如果你真的遇到這個問題，可以在**網站根目錄**的 `web.config` 檔案 (如果沒有這個檔案可以自己建立同名檔案) 加入一條 Rewrite 規則，讓 `.well-known/acme-challenge/IKibDaF4-FHZoGw1U6JTyGlBDMOtE-cQCFw13e4FaUc` 網址會直接**重寫**(Rewrite)到 `/acme-challenge.txt` 路徑，最後到網站根目錄建立一個 `acme-challenge.txt` 文字檔案，放入應該放入的內容即可：

Exercise

- 利用operator-sdk建立一個CRD與相對應的controller
  - CRD至少包含二個欄位
    - Image
    - NodePortNumber
  - Controller裡的Reconcile()和之前的Exercise做相同的事
    - 依CRD裡的Image建立Deployment,
    - 依CRD裡的NodePortNumber 建立類型為NodePort的service
  - CRD刪除時，同時刪除deployment與service
    - 透過設定OwnerReference達到
- 部署至K8S上執行

	HW-01	HW-02	HW-03	HW-04
主題	kubectl	client-go	informer	Operator-sdk
Watch 那個 Resource	--	--	Deployment	Web CRD
建立那些 Resource	Service Deployment	Service Deployment	Service	Service Deployment
Resource參數	同學自己定義在YAML內	同學自己定義在structure內	依測試用的 Deployment，產生適合的 Service	依測試用的 Web CRD，產生適合的 Service、Deployment

```
$ operator-sdk init --domain ntcu.edu.tw --repo github.com/example/sample
```

```
$ operator-sdk create api --group sample --version v1alpha1 --kind Sample --resource --controller
```

```
Image string `json:"image,omitempty"`
Key   string `json:"key,omitempty"`
Value string `json:"value,omitempty"`
```