第一次寫Operator就上手

莊家雋

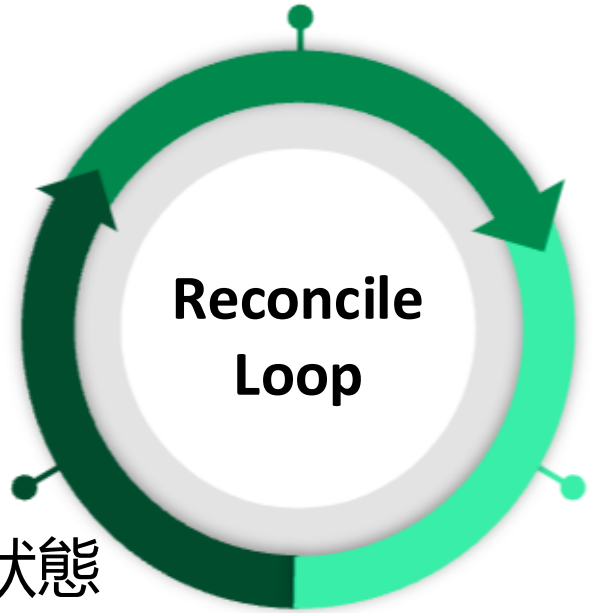
準備環境

https://github.com/ogre0403/K8S-Summit-2024-Operator101

預期狀態

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 6
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: my-image:latest
```

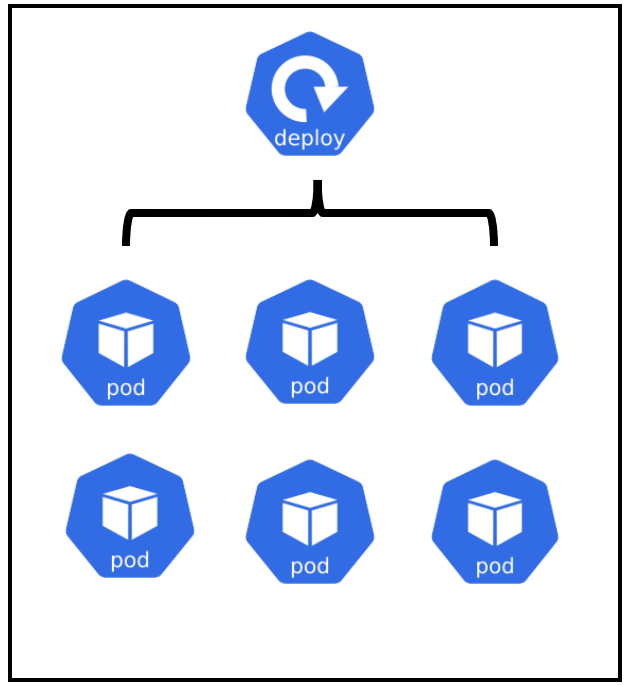
檢查當前狀態

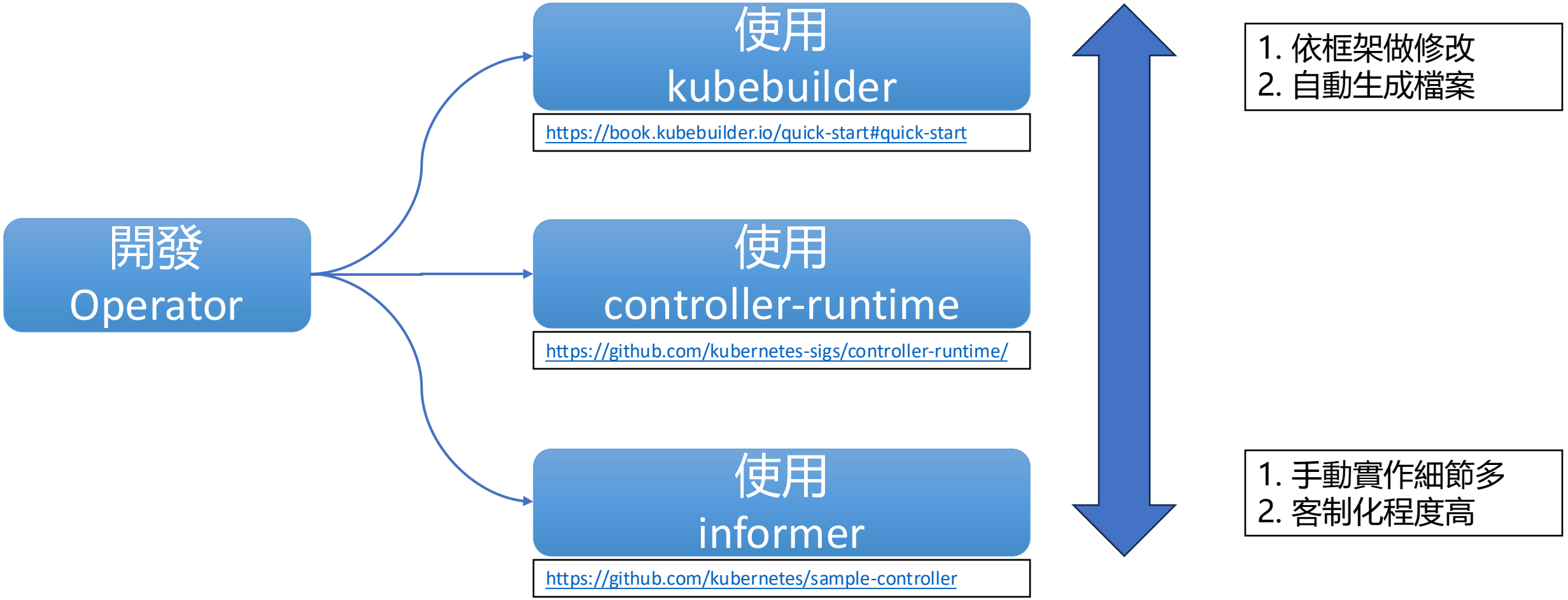


對比預期狀態

調整資源

真實狀態



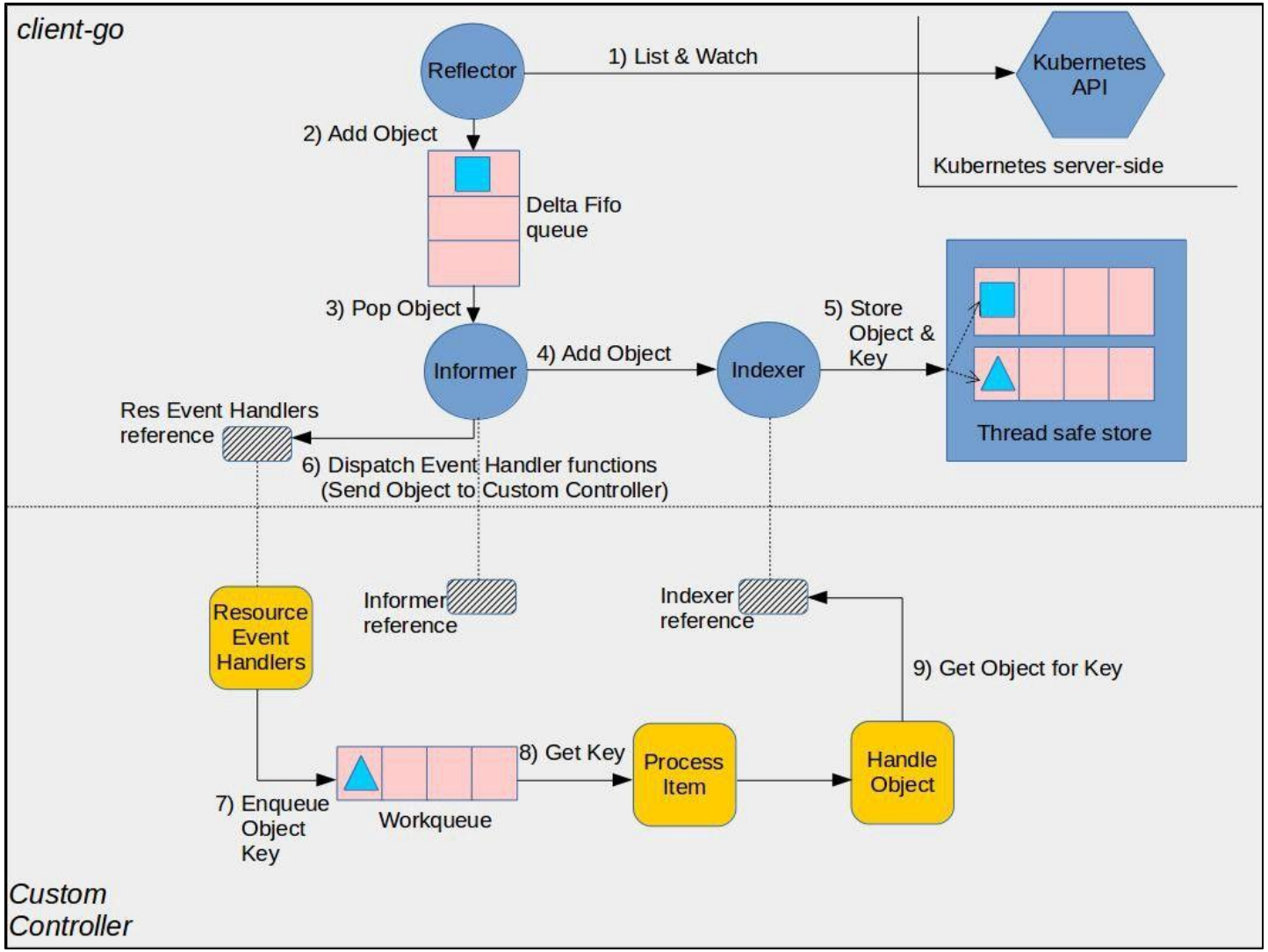

開發
Operator
使用
kubebuilder
https://book.kubebuilder.io/quick-start#quick-start
使用
controller-runtime
https://github.com/kubernetes-sigs/controller-runtime/
使用
informer
https://github.com/kubernetes/sample-controller
1. 依框架做修改
2. 自動生成檔案
1. 手動實作細節多
2. 客制化程度高

使用informer

sample-controller / controller.go

Code | Blame | 421 lines (372 loc) · 16.5 KB

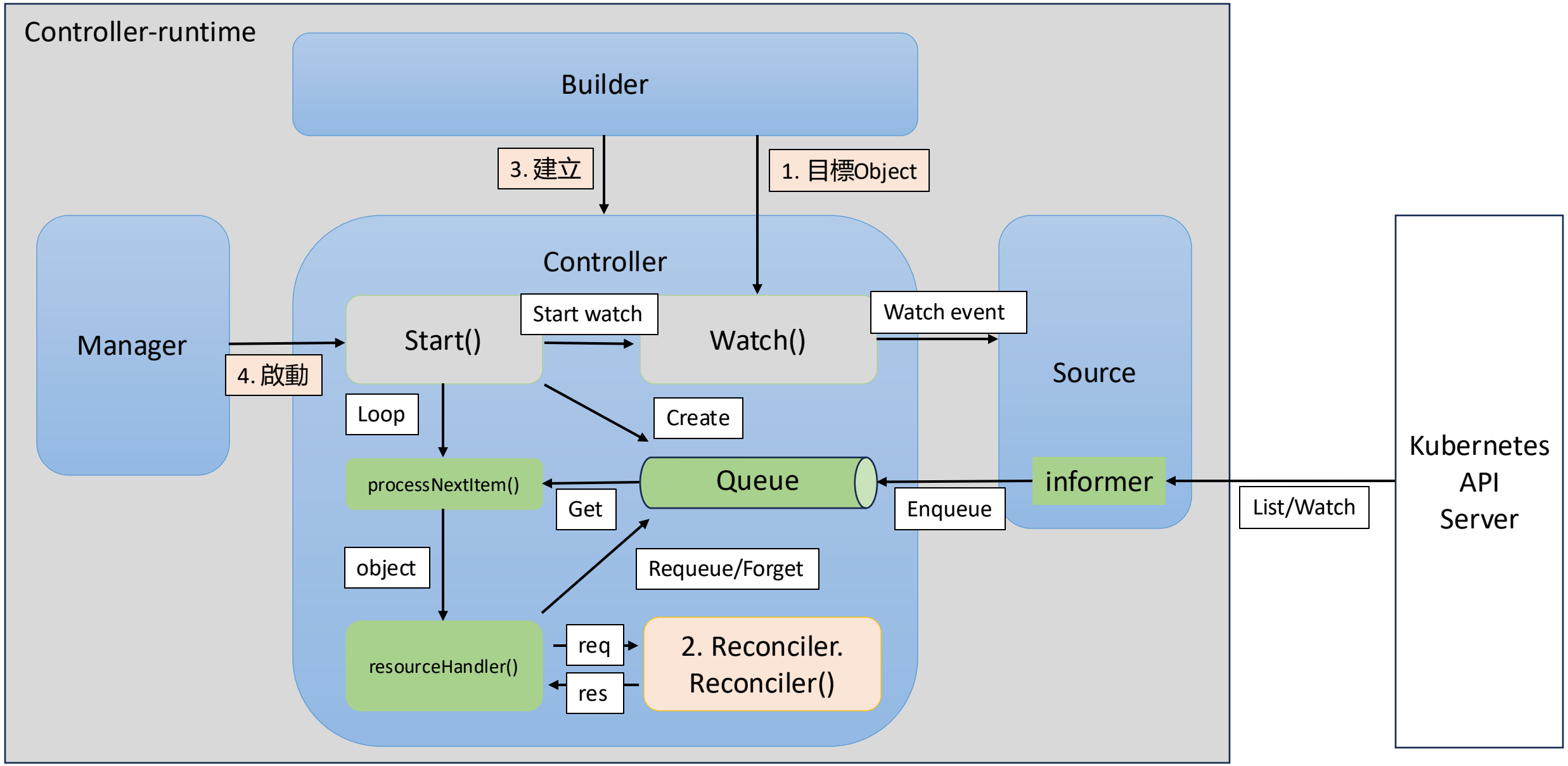
```
191  // processNextWorkItem function in order to read and process a message on the
192  // workqueue.
193  func (c *Controller) runWorker(ctx context.Context) {
194          for c.processNextWorkItem(ctx) {
195          }
196  }
197
198  // processNextWorkItem will read a single work item off the workqueue and
199  // attempt to process it, by calling the syncHandler.
200  func (c *Controller) processNextWorkItem(ctx context.Context) bool { ...
236  }
237
238  // syncHandler compares the actual state with the desired, and attempts to
239  // converge the two. It then updates the Status block of the Foo resource
240  // with the current status of the resource.
241  func (c *Controller) syncHandler(ctx context.Context, objectRef cache.ObjectName) error { ...
312  }
313
314  func (c *Controller) updateFooStatus(foo *samplev1alpha1.Foo, deployment *appsv1.Deployment) error { ...
326  }
327
328  // enqueueFoo takes a Foo resource and converts it into a namespace/name
329  // string which is then put onto the work queue. This method should *not* be
330  // passed resources of any type other than Foo.
331  func (c *Controller) enqueueFoo(obj interface{}) { ...
338  }
339
340  // handleObject will take any resource implementing metav1.Object and attempt
341  // to find the Foo resource that 'owns' it. It does this by looking at the
342  // objects metadata.ownerReferences field for an appropriate OwnerReference.
343  // It then enqueues that Foo resource to be processed. If the object does not
344  // have an appropriate OwnerReference, it will simply be skipped.
345  func (c *Controller) handleObject(obj interface{}) { ...
383  }
384
385  // newDeployment creates a new Deployment for a Foo resource. It also sets
386  // the appropriate OwnerReferences on the resource so handleObject can discover
387  // the Foo resource that 'owns' it.
388  func newDeployment(foo *samplev1alpha1.Foo) *appsv1.Deployment { ...
421  }
```



https://github.com/kubernetes/sample-controller/blob/master/controller.go

https://github.com/kubernetes/sample-controller/blob/master/docs/controller-client-go.md

使用Controller-runtime



使用kubebuilder

建立目錄

```
export CRD_NAME=myweb
export GROUP=operator.k8s-summit.org
export VERSION=v1
export BASE_PATH=operator

mkdir -p ${BASE_PATH}
cd ${BASE_PATH}
go mod init ${BASE_PATH}

mkdir hack
touch hack/boilerplate.go.txt

mkdir -p pkg/apis/${CRD_NAME}/${VERSION}
```

建立types.go 與 doc.go

```
# 建立 doc.go

cat << EOF > pkg/apis/${CRD_NAME}/${VERSION}/doc.go
// +k8s:deepcopy-gen=package
// +groupName=${GROUP}
package ${VERSION}
EOF
```

```
# 建立 types.go

cat << EOF > pkg/apis/${CRD_NAME}/${VERSION}/types.go
package ${VERSION}

import (
        metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)
```

Kubebuilder 提供指令初始化operator專案

Create a Project

Create a directory, and then run the init command inside of it to initialize a new project. Follows an example.

```
mkdir -p ~/projects/guestbook
cd ~/projects/guestbook
kubebuilder init --domain my.domain --repo my.domain/guestbook
```

Kubebuilder 提供指令初始化Resource Group/Version

Create an API

Run the following command to create a new API (group/version) as `webapp/v1` and the new Kind(CRD) `Guestbook` on it:

```
kubebuilder create api --group webapp --version v1 --kind Guestbook
```

使用kubebuilder

K8S-Summit-2024-Operator101 / 01-CRD / manifest / **crd-with-cols.yaml**

ogre0403 update CRD

Code | Blame | 54 lines (53 loc) · 1.17 KB

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: mywebs.operator.k8s-summit.org
spec:
  group: operator.k8s-summit.org
  scope: Namespaced
  names:
    plural: mywebs
    singular: myweb
    shortNames:
    - web
    kind: MyWeb
    categories:
    - all
  versions:
  - name: v1
    served: true
    storage: true
    subresources:
      status: {}
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
```

Kubebuilder 透過程式碼裡的annotation，可以自動生成YAML檔

If you are editing the API definitions, generate the manifests such as Custom Resources (CRs) or Custom Resource Definitions (CRDs) using

```
make manifests
```

▼ Click here to see an example. (`api/v1/guestbook_types.go`)

```go
// GuestbookSpec defines the desired state of Guestbook
type GuestbookSpec struct {
    // INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
    // Important: Run "make" to regenerate code after modifying this file

    // Quantity of instances
    // +kubebuilder:validation:Minimum=1
    // +kubebuilder:validation:Maximum=10
    Size int32 `json:"size"`

    // Name of the ConfigMap for GuestbookSpec's configuration
    // +kubebuilder:validation:MaxLength=15
    // +kubebuilder:validation:MinLength=1
    ConfigMapName string `json:"configMapName"`

    // +kubebuilder:validation:Enum=Phone;Address;Name
    Type string `json:"alias,omitempty"`
}
```

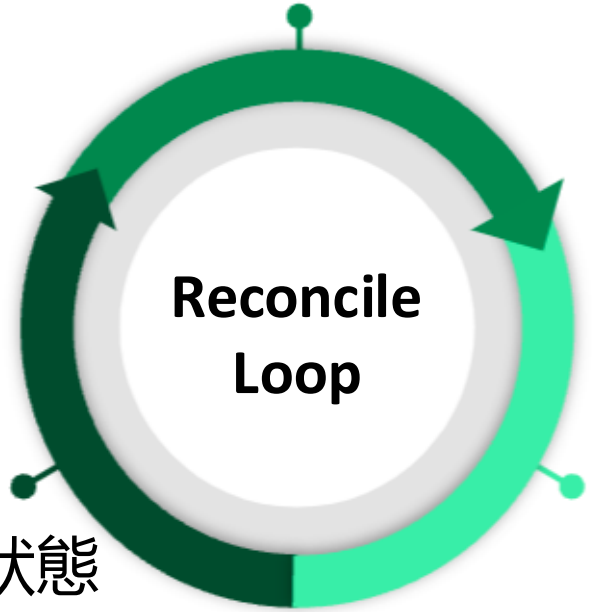
今天的目的...

- 設計 MyWeb CRD
- 生成 Customized Resource（CR) API
- 開發 MyWeb Operator
- 部署 Operator

預期狀態

```
apiVersion: operator.k8s-summit.org/v1
kind: MyWeb
metadata:
  name: myweb
  namespace: default
spec:
  image: nginx
  nodePortNumber: 30100
  pageContentHtml: |
    <html>
      <body>
        <h1>Hello, World!</h1>
      </body>
    </html>
```

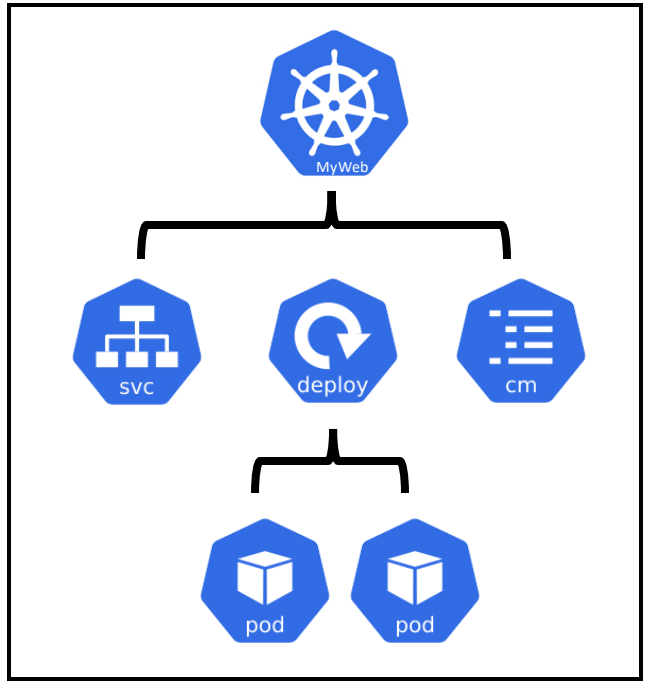
檢查當前狀態



對比預期狀態

調整資源

真實狀態



設計CRD

Custom Resource Definition

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: mywebs.operator.k8s-summit.org
spec:
  group: operator.k8s-summit.org
  scope: Namespaced
  names:
    plural: mywebs
    singular: myweb
    shortNames:
    - web
    kind: MyWeb
    categories:
    - all
  versions:
  - name: v1
    served: true
    storage: true
    subresources:
      status: {}
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              image:
                type: string
              nodePortNumber:
                type: integer
              pageContentHtml:
                type: string
```

Custom Resource

```yaml
apiVersion: operator.k8s-summit.org/v1
kind: MyWeb
metadata:
  name: myweb
  namespace: default
spec:
  image: nginx
  nodePortNumber: 30100
  pageContentHtml: |
    <html>
      <body>
        <h1>Hello, World!</h1>
      </body>
    </html>
```

生成CR API

- 建立CRD後，可以利用kubectl 建立 Custom Resource
- 但目前沒有任何的Go Package 可以處理Custom Resource

- 對每一個內建的Resource，Go Package都有提供相對應的clientset、informer、lister操作resource

- 對Custom Resource，可透過code-generator對Custom Resource生成clientset、informer、lister

安装 Code Generator

Install Code Generator

```
install-client-gen:
    go install k8s.io/code-generator/cmd/client-gen@v0.29.2

install-deepcopy-gen:
    go install k8s.io/code-generator/cmd/deepcopy-gen@v0.29.2

install-register-gen:
    go install k8s.io/code-generator/cmd/register-gen@v0.29.2

install-informer-gen:
    go install k8s.io/code-generator/cmd/informer-gen@v0.29.2

install-lister-gen:
    go install k8s.io/code-generator/cmd/lister-gen@v0.29.2
```

Generate by Code Generator

```
generate-deepcopy: install-deepcopy-gen
    deepcopy-gen \
    --input-dirs $(BASE_PATH)/pkg/apis/$(CRD_NAME)/$(VERSION) \
    -O zz_generated.deepcopy \
    --output-base .. \
    --go-header-file \
    ./hack/boilerplate.go.txt

generate-clientset: install-client-gen
    client-gen \
    --clientset-name clientset \
    --input-base "" \
    --input $(BASE_PATH)/pkg/apis/$(CRD_NAME)/$(VERSION) \
    --output-package $(BASE_PATH)/pkg/ \
    --output-base .. \
    --go-header-file ./hack/boilerplate.go.txt


generate-register: install-register-gen
    register-gen \
    -O zz_generated.register \
    --go-header-file ./hack/boilerplate.go.txt \
    --input-dirs ${BASE_PATH}/pkg/apis/${CRD_NAME}/${VERSION} \
    --output-base ..
```

doc.go

```go
// +k8s:deepcopy-gen=package
// +groupName=operator.k8s-summit.org
package v1
```

- doc.go 有 Group 與 Version資訊
- types.go 有Custom Resource 欄位的定義
- 使用code-generator生成API

```
.
├── Makefile
├── go.mod
├── go.sum
├── hack
│   └── boilerplate.go.txt
└── pkg
    └── apis
        └── myweb
            └── v1
                ├── doc.go
                └── types.go
```

types.go

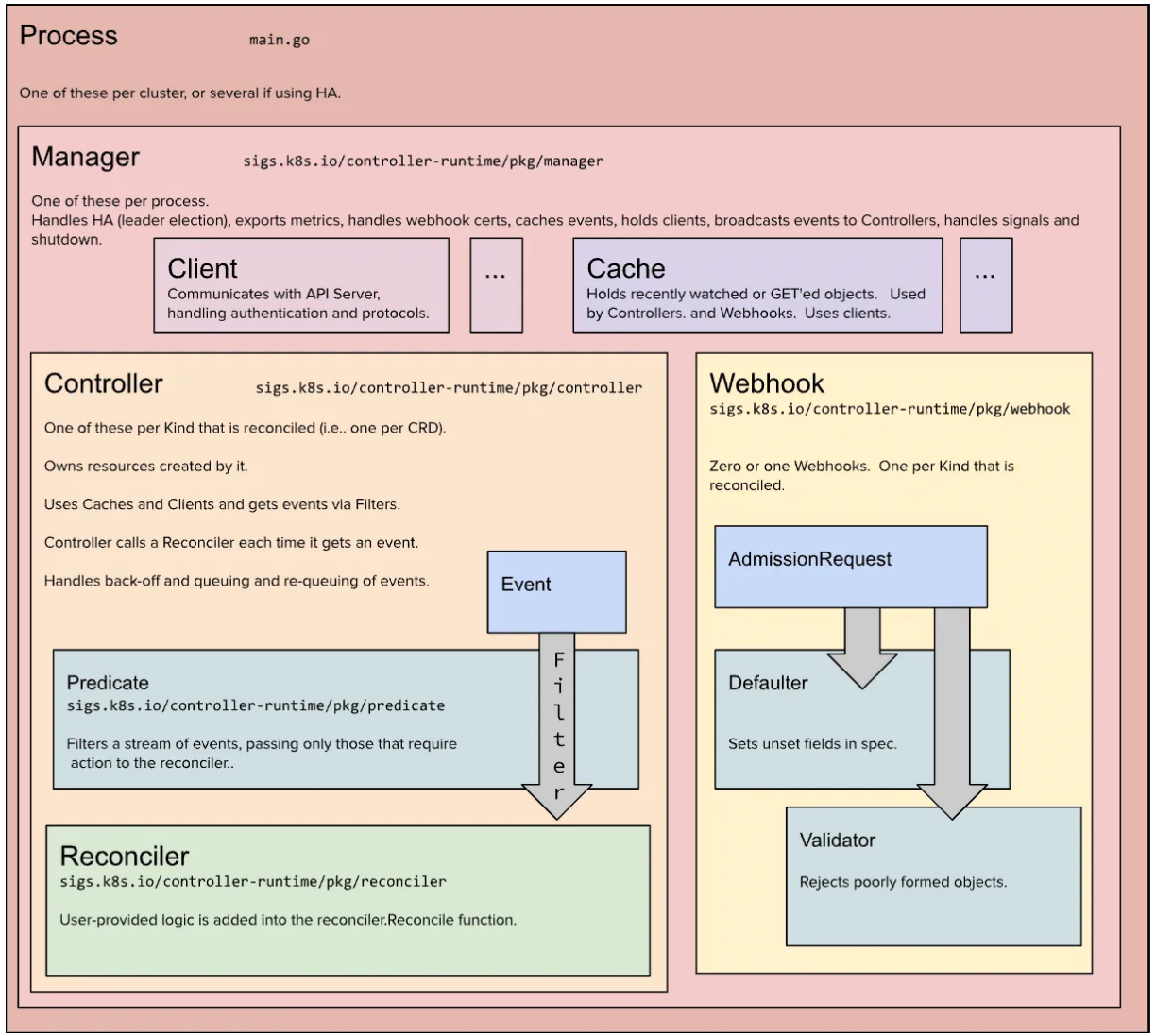
```go
package v1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
// +genclient
type MyWeb struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   MyWebSpec   `json:"spec"`
    Status MyWebStatus `json:"status"`
}

type MyWebSpec struct {
    Image           string `json:"image"`
    NodePortNumber  int    `json:"nodePortNumber"`
    PageContentHtml string `json:"pageContentHtml"`
}

type MyWebStatus struct {
    Completed bool `json:"completed"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type MyWebList struct {
    metav1.TypeMeta `json:",inline"`
    metav1.ListMeta `json:"metadata,omitempty"`

    Items []MyWeb `json:"items"`
}
```

Code Generate

撰寫Operator

Controller-runtime 主要元件

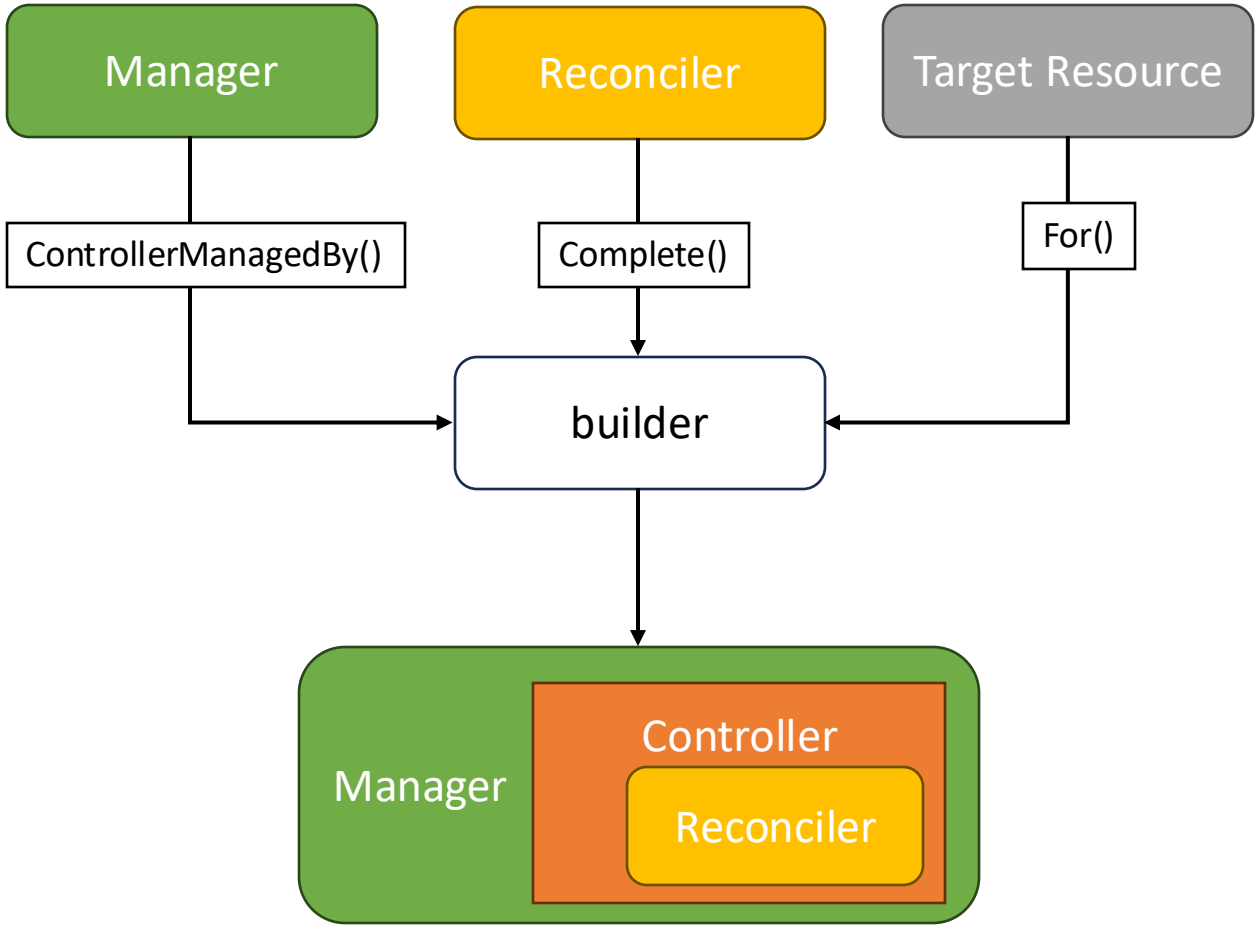
- Manager
  - 管理一群Controllers的life Cycle
  - 管理Controllers間的共用資源
- Controller
  - 透過K8S API 監控Target resource狀態的變化
  - 呼叫Reconciler.Reconcile()
- Reconciler
  - 存在Controller內部
  - 對CR的操作，都在Reconcile()裡



https://book.kubebuilder.io/architecture

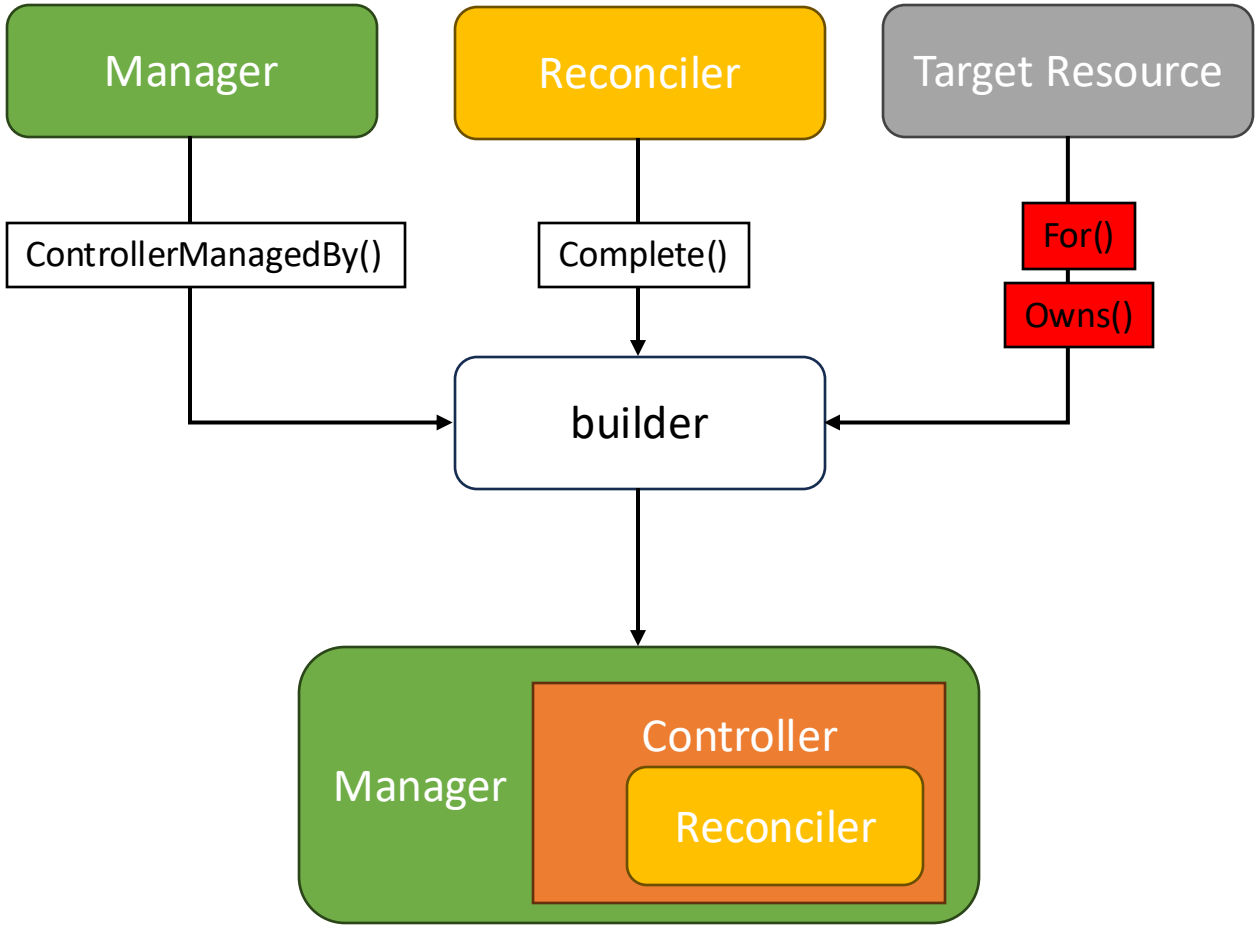
利用Builder 建立Controller

```
err = builder.
    ControllerManagedBy(mgr).
    For(&webv1.MyWeb{}).
    Complete(&MyReconciler{})
```



利用Builder 建立Controller

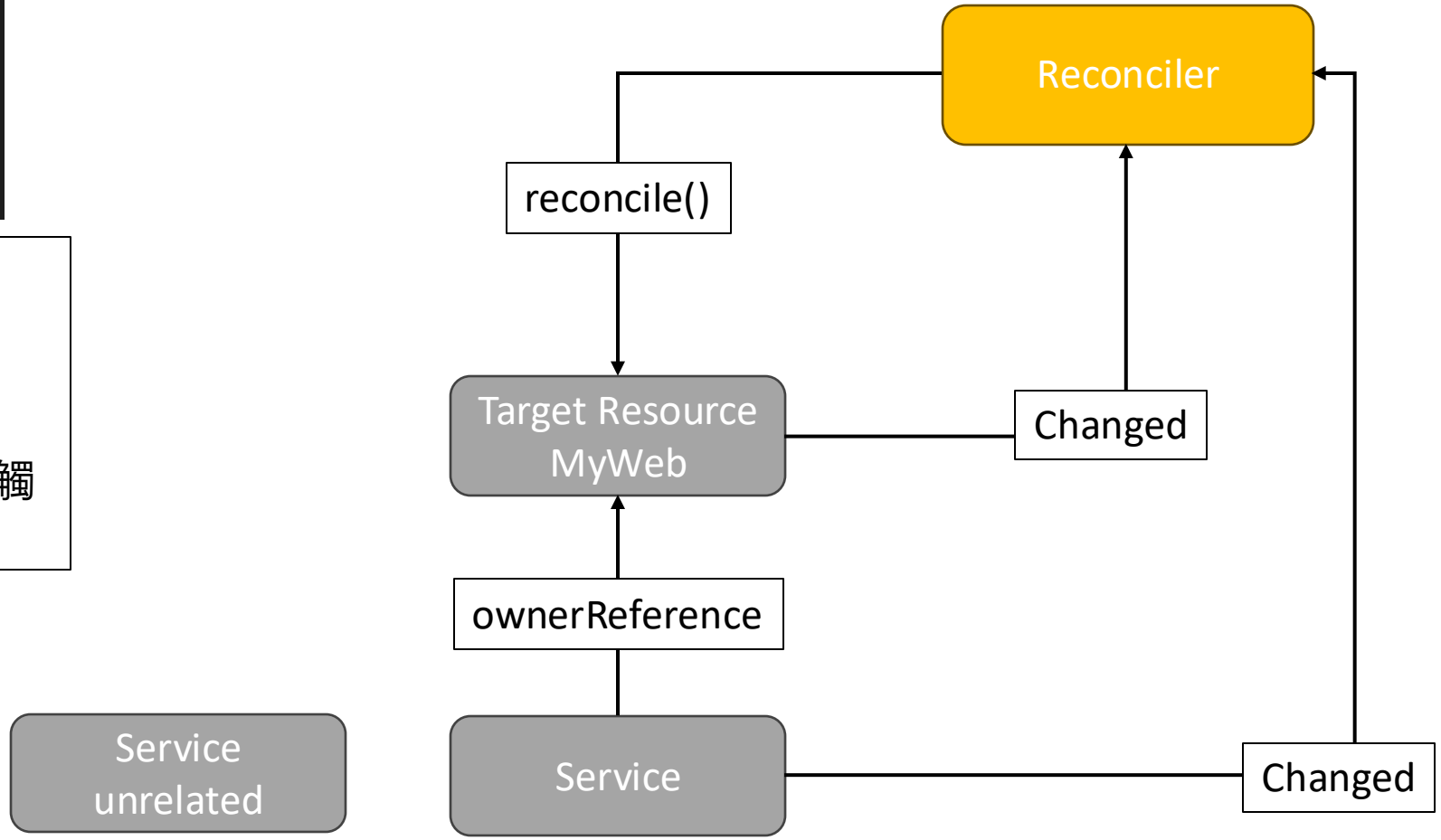
```
err = builder.
    ControllerManagedBy(mgr).
    For(&webv1.MyWeb{}).
    Owns(&corev1.ConfigMap{}).
    Owns(&corev1.Service{}).
    Owns(&appsv1.Deployment{}).
    Complete(&MyReconciler{})
```



```
err = builder.
    ControllerManagedBy(mgr).
    For(&webv1.MyWeb{}).
    Owns(&corev1.ConfigMap{}).
    Owns(&corev1.Service{}).
    Owns(&appsv1.Deployment{}).
    Complete(&MyReconciler{})
```

For():
Target Resource改變，觸發Reconcile()

Owns():
Target Resource擁有的Resource變動，也會觸發Reconcile()

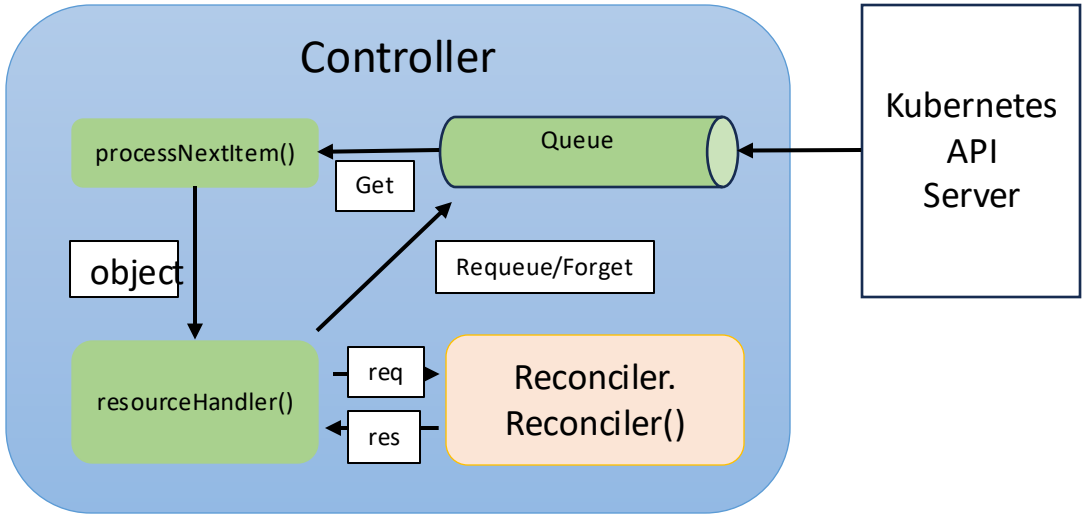


Reconciler

```go
func (r *WebReconciler) Reconcile(ctx context.Context, req reconcile.Request) (reconcile.Result, error) {

    log := log.FromContext(ctx)

    sample := &webv1.MyWeb{}
    err := r.client.Get(ctx, req.NamespacedName, sample)
    if err != nil {…
    }

    foundCM := &corev1.ConfigMap{}
    err = r.client.Get(ctx, types.NamespacedName{Name: sample.Name, Namespace: sample.Namespace}, foundCM)
    if err != nil && errors.IsNotFound(err) {…
    } else if err != nil {…
    }

    foundDeployment := &appsv1.Deployment{}
    err = r.client.Get(ctx, types.NamespacedName{Name: sample.Name, Namespace: sample.Namespace}, foundDeployment
    if err != nil && errors.IsNotFound(err) {…
    } else if err != nil {…
    }

    foundSvc := &corev1.Service{}
    err = r.client.Get(ctx, types.NamespacedName{Name: sample.Name, Namespace: sample.Namespace}, foundSvc)
    if err != nil && errors.IsNotFound(err) {…
    } else if err != nil {…
    }

    // Update PageContentHtml and NodePortNumber

    html := sample.Spec.PageContentHtml
    if foundCM.Data["index.html"] != html {…
    }

    nodePort := sample.Spec.NodePortNumber
    if foundSvc.Spec.Ports[0].NodePort != int32(nodePort) {…
    }

    log.Info("Exiting Reconcile")

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

建立

更新



- Reconciler 不知道是什麼原因執行 Reconciler()
  - 需要保持 Idempotent
- 回傳Error觸發retry

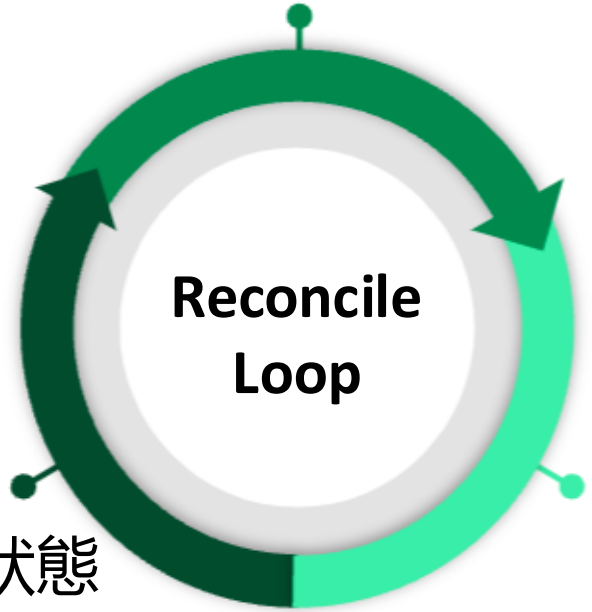
部署

- Build Operator Image
- Operator manifest
  - Reconciler對MyWeb、Service、Deployment、ConfigMap有操作
  - 必需讓Operator有相對應的Role

預期狀態

```
apiVersion: operator.k8s-summit.org/v1
kind: MyWeb
metadata:
  name: myweb
  namespace: default
spec:
  image: nginx
  nodePortNumber: 30100
  pageContentHtml: |
    <html>
      <body>
        <h1>Hello, World!</h1>
      </body>
    </html>
```

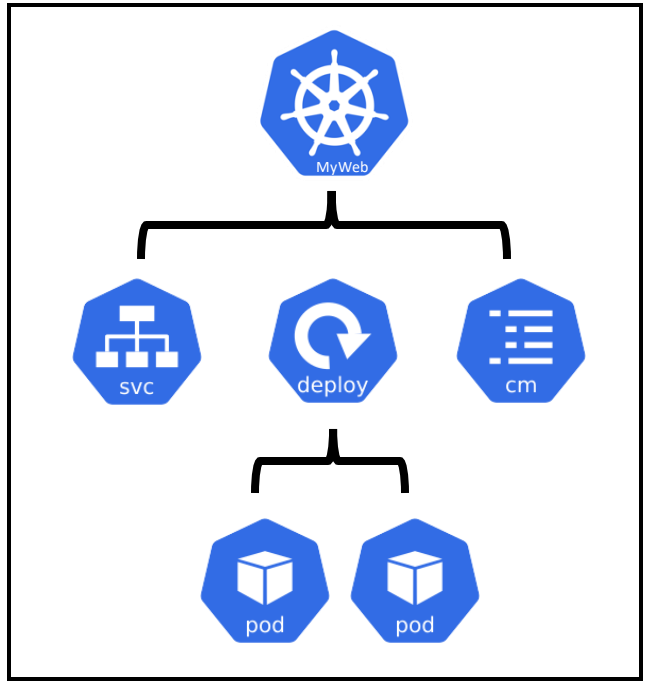
檢查當前狀態



對比預期狀態

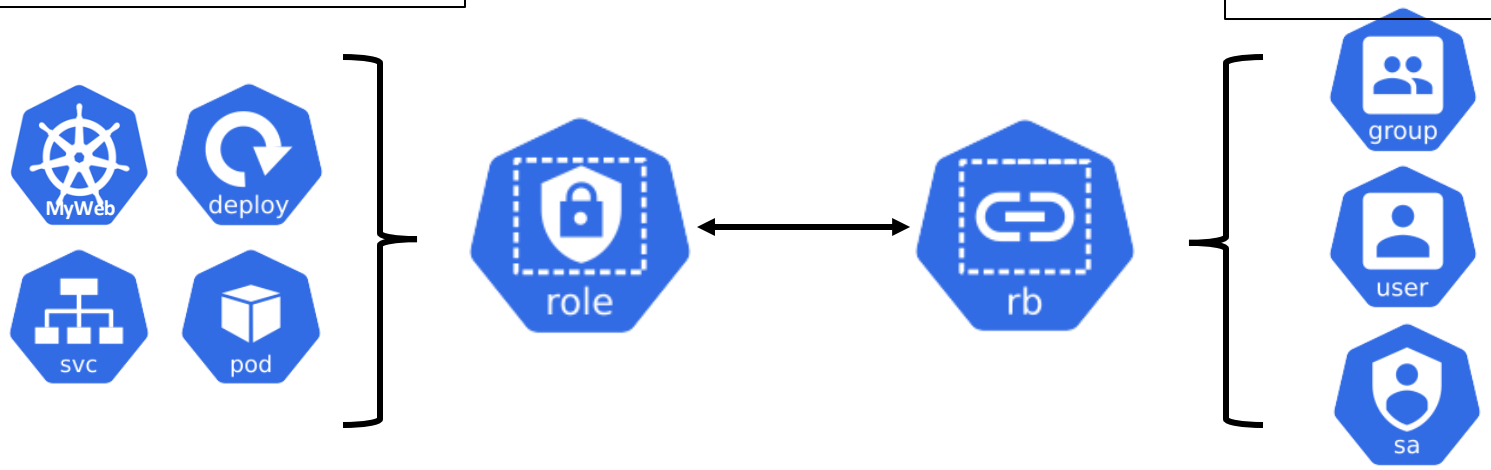
調整資源

真實狀態



對operator會使用的resource給與權限

將Role綁定給Service Account




```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata: …
rules:
  - apiGroups:
      - apps
    resources:
      - deployments
    verbs: …
  - apiGroups:
      - ""
    resources:
      - configmaps
      - services
    verbs: …
  - apiGroups:
      - operator.k8s-summit.org
    resources:
      - mywebs
    verbs: …
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: operator-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: operator-role
subjects:
  - kind: ServiceAccount
    name: operator
    namespace: default
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: operator
spec:
  replicas: 1
  selector:
    matchLabels:
      app: operator
  template:
    metadata:
      labels:
        app: operator
    spec:
      serviceAccountName: operator
      containers:
        - name: operator
          image: operator:latest
          imagePullPolicy: Never
```

總結

- 設計 MyWeb CRD
- 生成 Customized Resource (CR) API
- 開發 MyWeb Operator
- 部署 Operator