

Kubernetes Operator Pattern 實作

莊家雋

什麼是Kubernetes

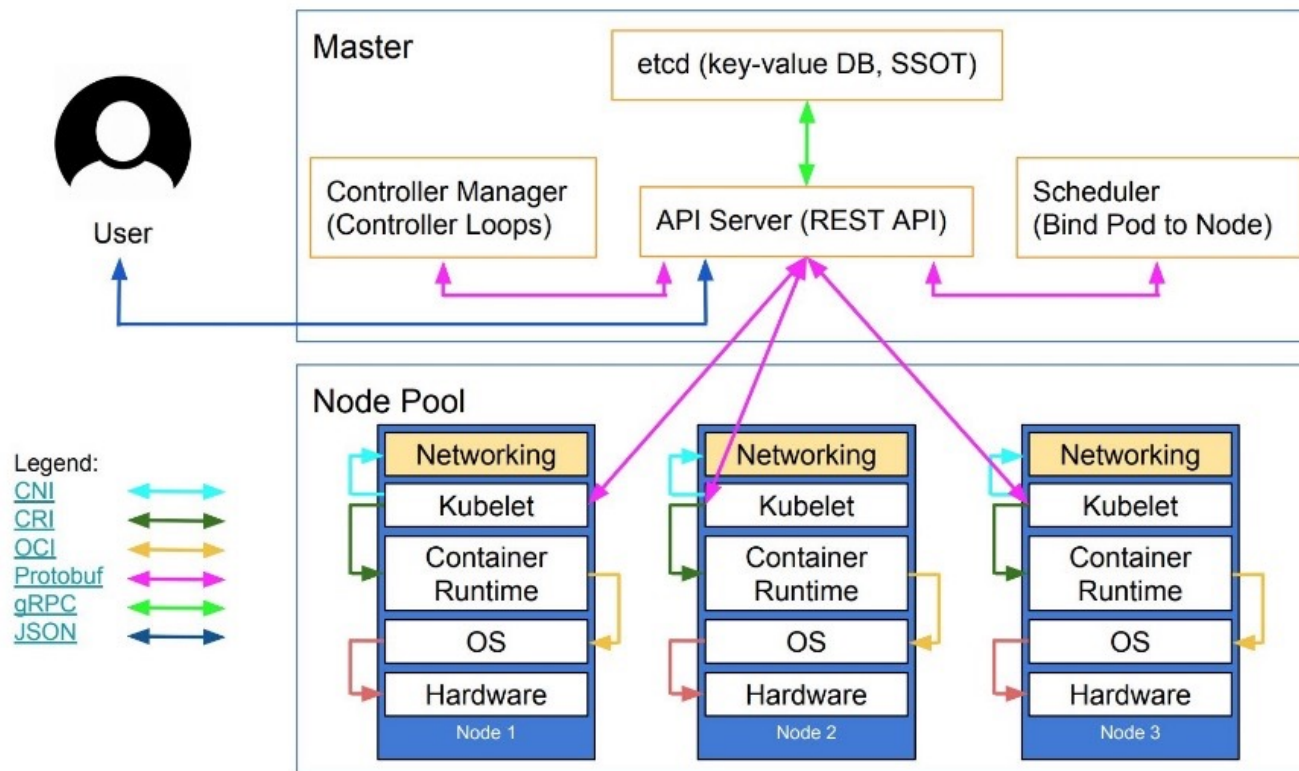
- Kubernetes 是 Google 開源的Container分散式管理系統
- 管理OCI標準映像檔的容器叢集排程服務，簡稱為k8s('k' + 8 letters + 's')



Kubernetes 基本架構

- Kubernetes 屬於分散式架構系統，主要由master與多個node組成

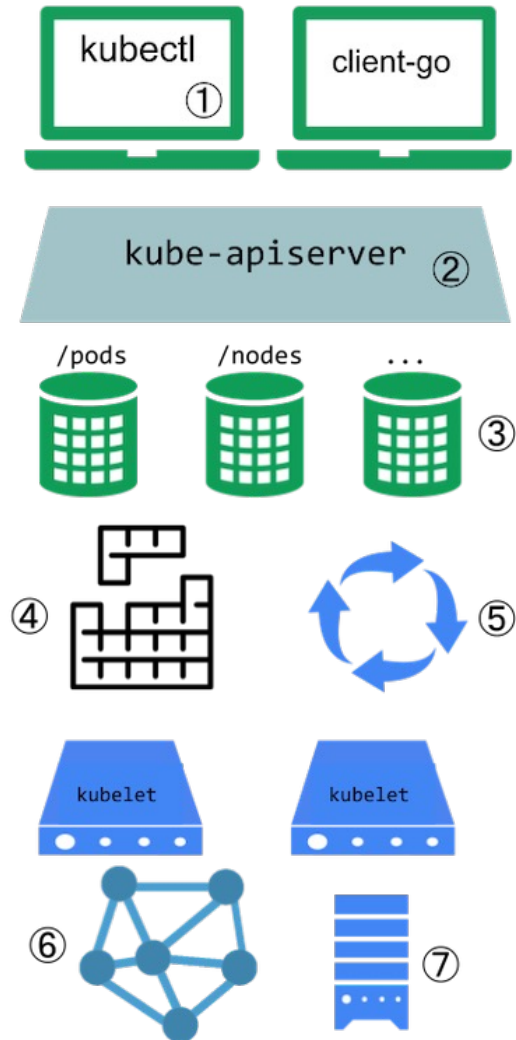
Kubernetes' high-level component architecture



Kubernetes 基本機制

- configuration convergence:
 - 不斷的比較期望的配置和實際的配置，修訂實際配置以收斂到期望配置
- 寫入期望配置：
 - Kubernetes 裡的所有資源對象，Service、Deployment、等等，都是通過 api-server 檢查格式後，序列化並存入 etcd
- 收斂到期望配置：
 - controller 使用 api-server 的 watch API 收取 etcd 裡資源的期望配置，和通過 kubelet 收集到的實際配置做對比並修正差異

Kubernetes Extension Point



- 1. Kubectl Plugin
- 2. Admission Control hook
- 3. Custom Resource
- 4. Scheduler
- 5. Custom Resource Controller
- 6. CNI
- 7. Device Plugin

Operator Pattern

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.

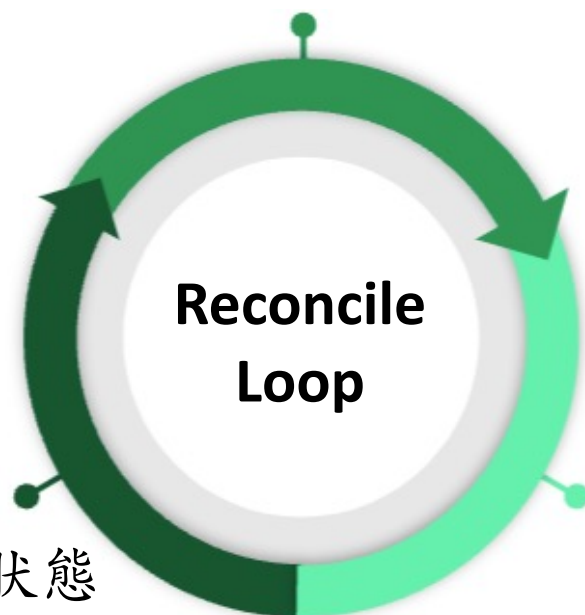
People who run workloads on Kubernetes often like to use automation to take care of repeatable tasks. The Operator pattern captures how you can write code to automate a task beyond what Kubernetes itself provides.

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

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
```

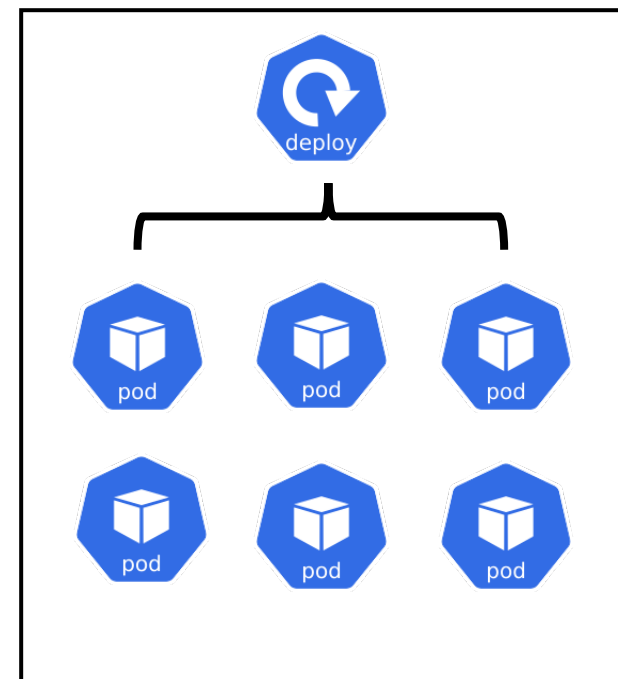
檢查當前狀態



對比預期狀態

調整資源

真實狀態



Case Study: Cert manager

• Let's Encrypt HTTP-01 Challenge

```
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-FHZoGw1U6JTyG1BDM0tE-cQCFw13e4FaUc`

檔案內容：

`IKibDaF4-FHZoGw1U6JTyG1BDM0tE-cQCFw13e4FaUc.plEmWe4UXqKWJvuRWXDnZDtkeEh2omjTeQWuZ`

Create a file containing just this data:

`IKibDaF4-FHZoGw1U6JTyG1BDM0tE-cQCFw13e4FaUc.plEmWe4UXqKWJvuRWXDnZDtkeEh2omjTeQWuZ
HEKan4`

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

`http://angular.tw/.well-known/acme-challenge/IKibDaF4-FHZoGw1U6JTyG1BDM0tE-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-FHZoGw1U6JTyG1BDM0tE-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：

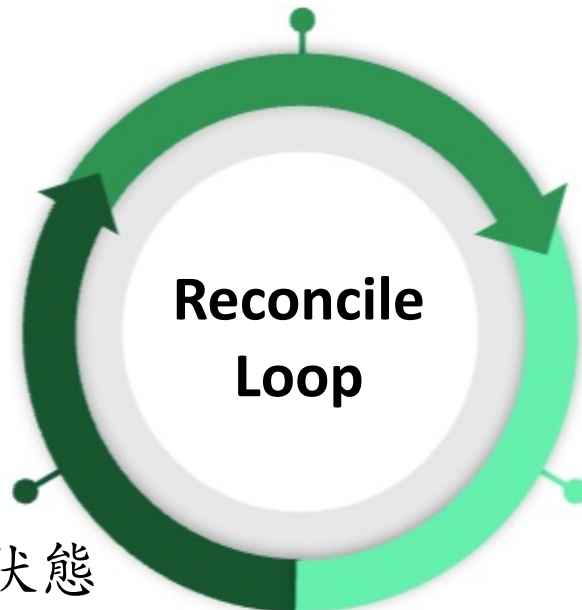
今天的目的...

- 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>
```

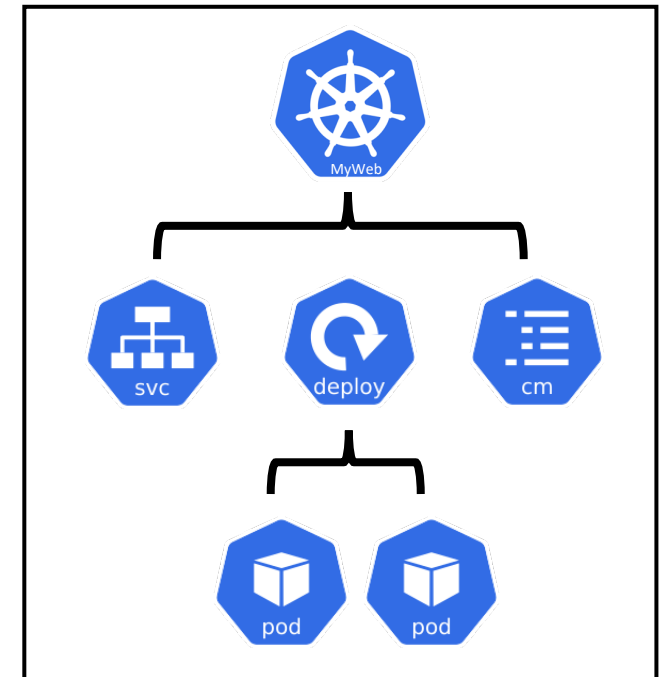
檢查當前狀態



對比預期狀態

調整資源

真實狀態



Operator Pattern

- 設計 CRD
 - 定義定制化的CR
 - 生成 Customized Resource (CR) API
- 開發 Operator
 - Reconcile邏輯
 - CR建立後，依照業務邏輯進行處理
 - CR變動後，依照業務邏輯進行處理
 - CR刪除後，依照業務邏輯進行處理
- 部署 Operator 到 K8S

Custom Resource (CR) and Custom Resource Definition (CRD)

Outline

- Custom Resource (CR)
- Custom Resource Definition (CRD)
- Define your own CRD
- CR的go-client

Custom Resource

- Resource
 - K8S預設安裝後即有的Resource, eg: Pod 、 Node 、 ...
- Custom Resource
 - 擴充K8S的一種機制，讓K8S得以支援預設之外的Resource
 - 可以動態註冊新的Custom Resource讓K8S使用，而不需動新安裝或啟動
- Controller
 - Custom Resource宣告想要得到的現象
 - 需透過相對應的Controller，完成想要實現的工作

Custom Resource Definition

- K8S內建的一個Resource，和Pod、Node...相同
- 用來定義 Custom Resource的Schema
 - CR的CRD不存在時，建立CR會失敗。
- 可以用YAML寫CRD，也可以用go-client寫CRD

Custom Resource “Project ”

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

```
apiVersion:
  apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: projects.xyz.com
spec:
  group: xyz.com
  version: v1
  scope: Namespaced
  names:
    plural: projects
    singular: project
    kind: Project
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                image:
                  type: string
                password:
                  type: string
            status:
              type: object
              properties:
                Ready:
                  type: boolean
```

Custom Resource Definition for Custom Resource “Project ”

CR & CRD 範例


```
apiVersion: apiextensions.k8s.io/v1
```

```
kind: CustomResourceDefinition
```

```
metadata:
```

```
  name: myresources.mygroup.example.com
```

The complete name of the new resource, including its group

```
spec:
```

```
  group: mygroup.example.com
```

The group the new resource belongs to

```
  scope: Namespaced
```

The new resource can be created in specific namespaces

```
  names:
```

```
    plural: myresources
```

```
    singular: myresource
```

```
    shortNames:
```

```
      - my
```

Short names of the new resource, you can use `kubectl get my`, `kubectl get myres`

```
      - myres
```

```
    kind: MyResource
```

The kind of the Custom resource

```
    categories:
```

```
      - all
```

```
  versions:
```

```
    - name: v1alpha1
```

v1alpha1 version is the only version defined for the new resource

```
      served: true
```

```
      storage: true
```

```
      schema:
```

```
        openAPIV3Schema:
```

```
          type: object
```

Defines the new resource schema as an object, with no field

```

versions:
- name: v1alpha1
  served: true
  storage: true
  subresources:
    status: {}
  schema:
    openAPIV3Schema:
      type: object
      properties:
        spec:
          type: object
          properties:
            image:
              type: string
            memory:
              x-kubernetes-int-or-string: true
        status:
          type: object
          properties:
            state:
              type: string
      additionalPrinterColumns:
        - name: image
          jsonPath: .spec.image
          type: string
        - name: memory
          jsonPath: .spec.memory
          type: string
        - name: age
          jsonPath: .metadata.creationTimestamp
          type: date

```

Spec and Status fields will also be of type **object** and contain properties.

The accepted data types are: string, number, integer, Boolean, array, object

Spec and Status fields will also be of type **object** and contain properties.

The **AdditionalPrinterColumns** field of the CRD spec is used to indicate which columns of the resource you want to be displayed in the output of `kubectl get <resource>`.

```

$ kubectl apply -f myres1.yaml
myresource.mygroup.example.com/myres1
$ kubectl get myresources
NAME      IMAGE    MEMORY    AGE
myres1    nginx    1024Mi    12m

```

Custom Resource Definition

```
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
```

Custom Resource

```
apiVersion: operator.k8s-summit.org/v1
kind: MyWeb
metadata:
  name: myweb
  namespace: default
spec:
  image: nginx
  nodePortNumber: 30100
```

CR的go-client

```
read, err := clientset.  
    CoreV1().  
    ConfigMaps(namespace).  
    Get(  
        context.Background(),  
        name,  
        metav1.GetOptions{},  
    )
```

```
read, err = clientset.  
    MygroupV1alpha1().  
    MyResources(namespace).  
    Get(  
        context.Background(),  
        "myresource",  
        metav1.GetOptions{}  
    )
```

- 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

流程

- 手動寫types.go 與 docs.go
 - 依CRD 定義的規格，完成types.go
 - 依CRD 定義的version與group，完成docs.go
- 安裝code-generator
- 下載相依性套件
- 自動生成clientset
 - 生成 deepcopy檔
 - 生成registry檔
 - 生成clientset package

安裝 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 ..
```

```
GO doc.go ×
1 // +k8s:deepcopy-gen=package
2 // +groupName=operator.k8s-summit.org
3 package v1
4
```

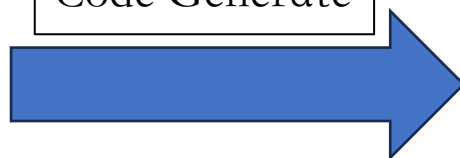
- doc.go 有 Group 與 Version 資訊
- types.go 有 Custom Resource 欄位的定義
- 使用code-generator生成API



```
GO types.go ×
1 package v1
2
3 import (
4     metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
5 )
6
7 // +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
8 // +genclient
9 type MyWeb struct {
10     metav1.TypeMeta `json:",inline"`
11     metav1.ObjectMeta `json:"metadata,omitempty"`
12
13     Spec   MyWebSpec   `json:"spec"`
14     Status MyWebStatus `json:"status"`
15 }
16
17 type MyWebSpec struct {
18     Image           string `json:"image"`
19     NodePortNumber int    `json:"nodePortNumber"`
20     PageContentHtml string `json:"pageContentHtml"`
21 }
22
23 type MyWebStatus struct {
24     Completed bool `json:"completed"`
25 }
26
27 // +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
28 type MyWebList struct {
29     metav1.TypeMeta `json:",inline"`
30     metav1.ListMeta `json:"metadata,omitempty"`
31
32     Items []MyWeb `json:"items"`
33 }
34
```



Code Generate



Operator程式開發

- 1.[Kubernetes Operator series 1 — controller-runtime example controller](#)
- 2.[Kubernetes Operator series 2 — Overview of controller-runtime](#)
- 3.[Kubernetes Operator series 3 — controller-runtime component — Manager](#)
- 4.[Kubernetes Operator series 4 — controller-runtime component — Builder](#)
- 5.[Kubernetes Operator series 5 — controller-runtime component — Reconciler](#)
- 6.[Kubernetes Operator series 6 — controller-runtime component — Controller](#)

Flow

- Create Manager
 - Build Controller
- Implement Reconcile()

Process

main.go

One of these per cluster, or several if using HA.

Manager

sigs.k8s.io/controller-runtime/pkg/manager

One of these per process.

Handles HA (leader election), exports metrics, handles webhook certs, caches events, holds clients, broadcasts events to Controllers, handles signals and shutdown.

Client

Communicates with API Server, handling authentication and protocols.

...

Cache

Holds recently watched or GET'ed objects. Used by Controllers. and Webhooks. Uses clients.

...

Controller

sigs.k8s.io/controller-runtime/pkg/controller

One of these per Kind that is reconciled (i.e.. one per CRD).

Owns resources created by it.

Uses Caches and Clients and gets events via Filters.

Controller calls a Reconciler each time it gets an event.

Handles back-off and queuing and re-queuing of events.

Event

Predicate

sigs.k8s.io/controller-runtime/pkg/predicate

Filters a stream of events, passing only those that require action to the reconciler..

F
i
l
t
e
r

Reconciler

sigs.k8s.io/controller-runtime/pkg/reconciler

User-provided logic is added into the reconciler.Reconcile function.

Webhook

sigs.k8s.io/controller-runtime/pkg/webhook

Zero or one Webhooks. One per Kind that is reconciled.

AdmissionRequest

Defaulter

Sets unset fields in spec.

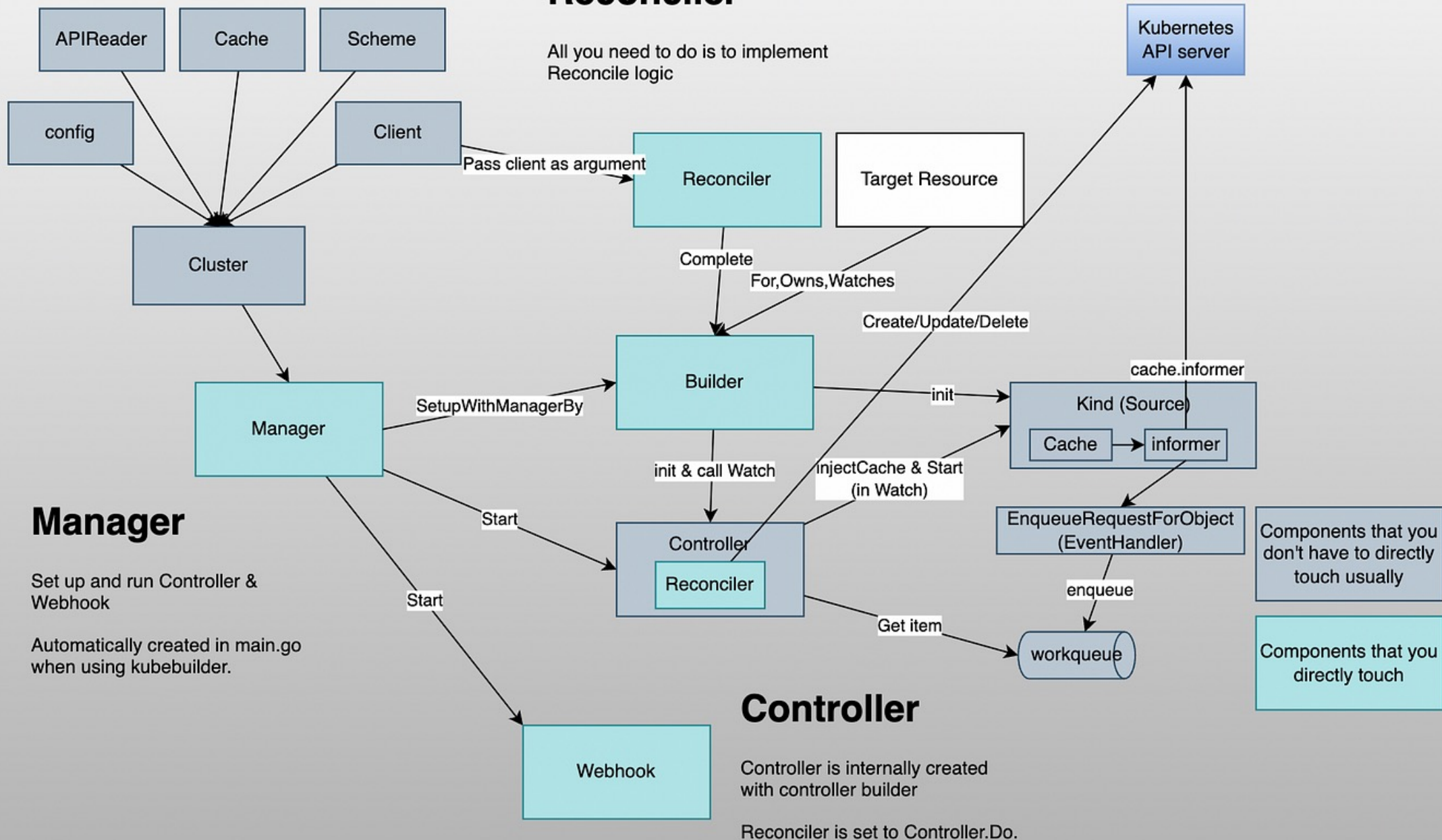
Validator

Rejects poorly formed objects.

Architecture

Reconciler

All you need to do is to implement Reconcile logic



```

func main() {

    mgr, err := manager.New(
        config.GetConfigOrDie(),
        manager.Options{
            Scheme: scheme,
        },
    )

    if err != nil {
        panic(err)
    }

    err = builder.
        ControllerManagedBy(mgr).
        For(&myresourcev1alpha1.MyResource{}).
        Owns(&corev1.ConfigMap{}).
        Owns(&batchv1.Job{}).
        Complete(&MyReconciler{})

    if err != nil {
        panic(err)
    }

    err = mgr.Start(context.Background())

    if err != nil {
        panic(err)
    }
}

```

```

type MyReconciler struct {
    client client.Client
    scheme *runtime.Scheme
}

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

    log := log.FromContext(ctx)

    sample := &myresourcev1alpha1.MyResource{}
    err := r.client.Get(ctx, req.NamespacedName, sample)
    if err != nil {
        ...
    }

    foundJob := &batchv1.Job{}
    err = r.client.Get(ctx, types.NamespacedName{Name: sample.Name, Namespace: sample.Namespace}, foundJob)
    if err != nil && errors.IsNotFound(err) {
        ...
    } else 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 {
        ...
    }

    return reconcile.Result{}, nil
}

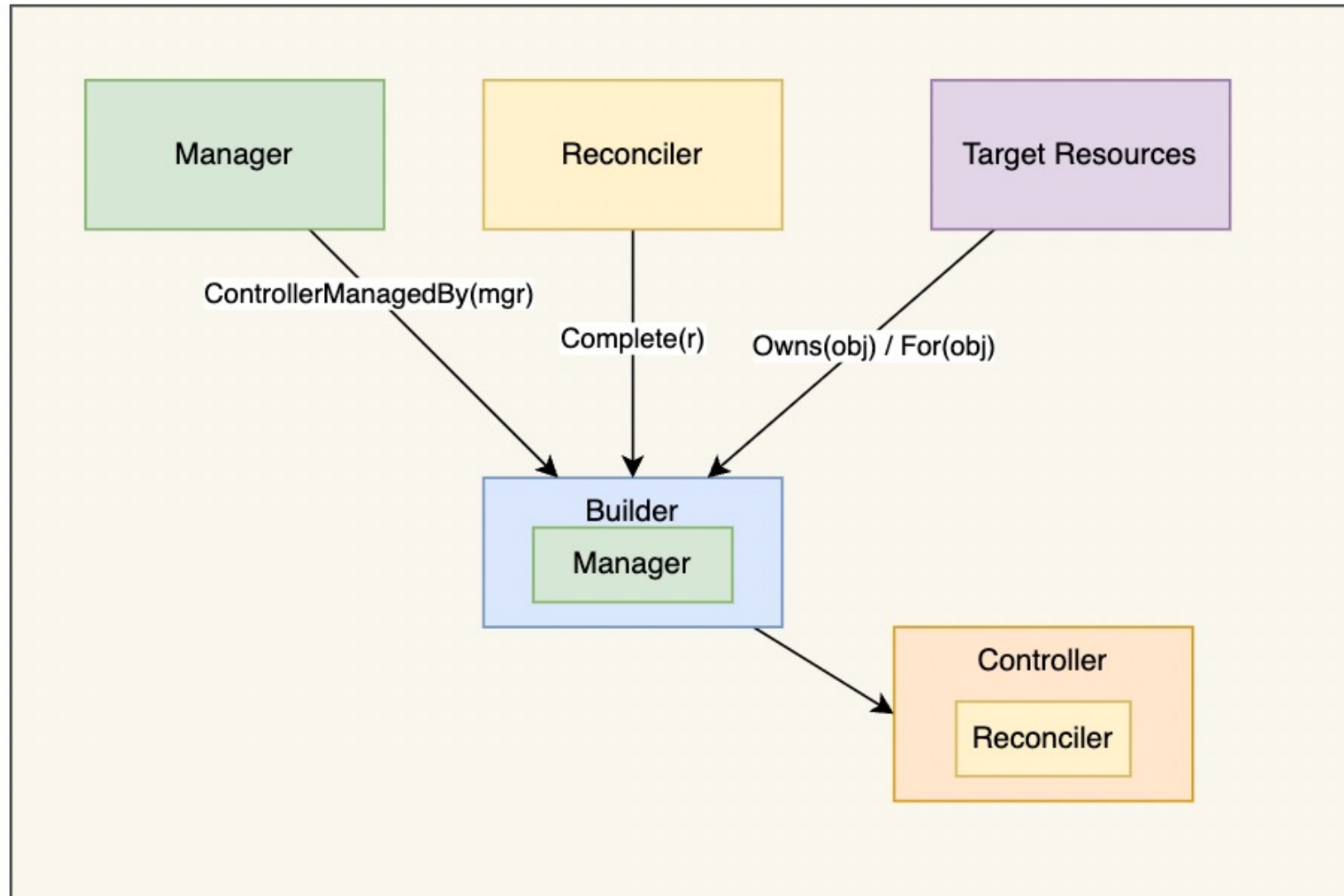
func (r *MyReconciler) newConfigMap(s *myresourcev1alpha1.MyResource) *corev1.ConfigMap {
    ...
}

func (r *MyReconciler) newJob(s *myresourcev1alpha1.MyResource) *batchv1.Job {
    ...
}

```

Core Components

- Manager
- Controller
- Reconciler
- Builder

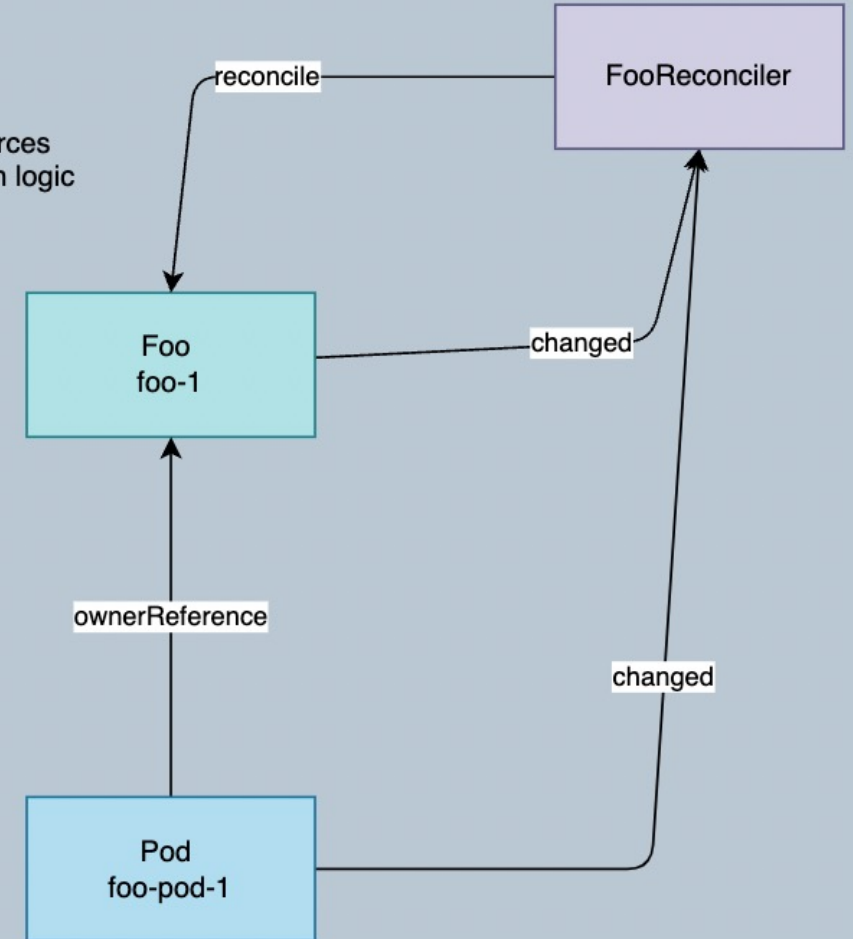


For & Owns

```
mgr, err := manager.New(  
    config.GetConfigOrDie(),  
    manager.Options{  
        Scheme: scheme,  
    },  
)  
  
if err != nil {  
    panic(err)  
}  
  
err = builder.  
    ControllerManagedBy(mgr).  
    For(&myresourcev1alpha1.MyResource{}).  
    Owns(&corev1.ConfigMap{}).  
    Owns(&batchv1.Job{}).  
    Complete(&MyReconciler{})
```

For

Change of the target resources
will trigger the reconciliation logic
against the object



Not owned

changes of unrelated object will
not trigger the reconciliation logic

Owns

Change of the owned objects will
trigger the reconciliation logic
against the owner

OwnerReference

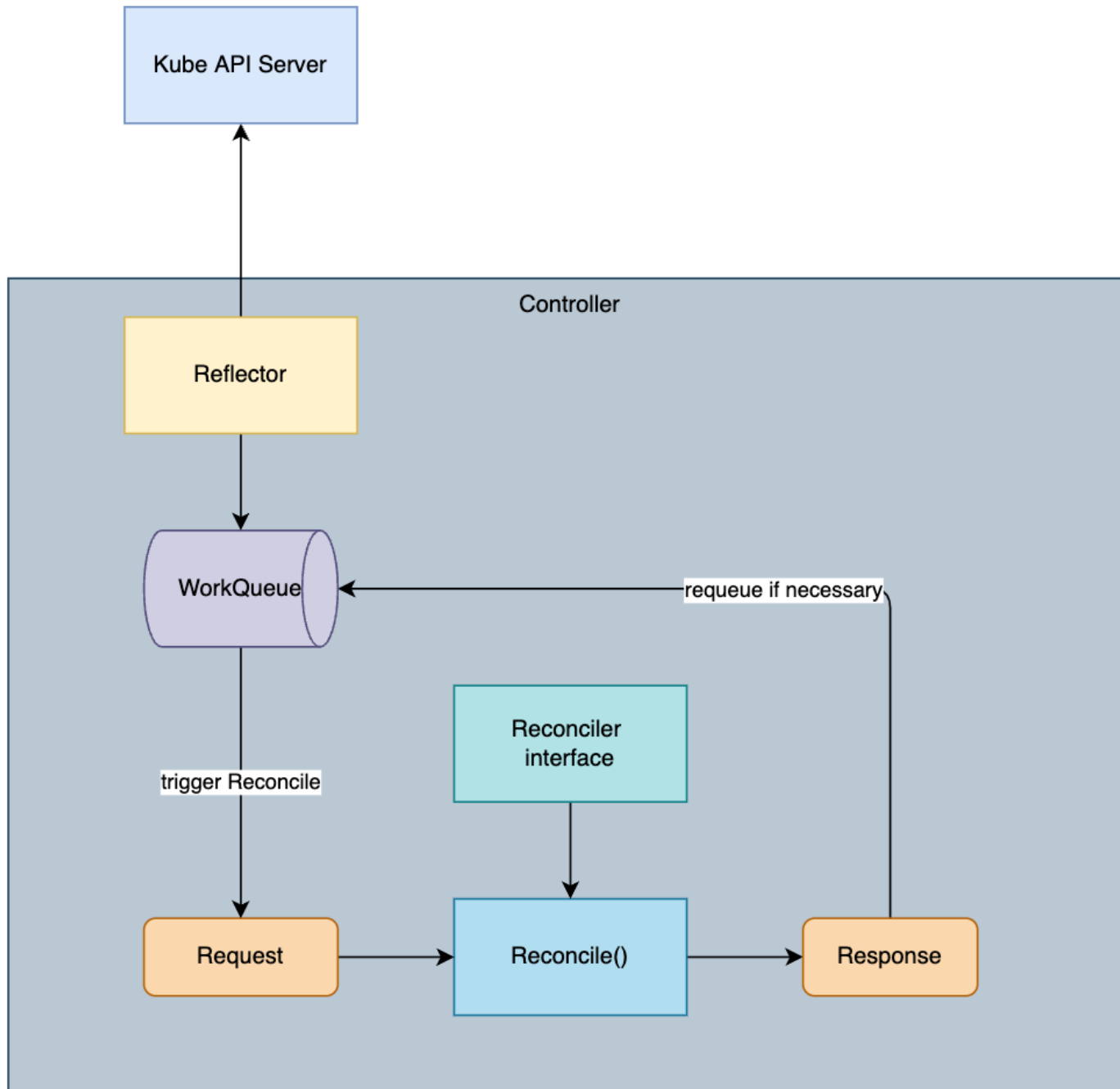
- 指定Owner 和 Dependent的關係
- 刪除Owner時， Dependent同時被刪除

```
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
```



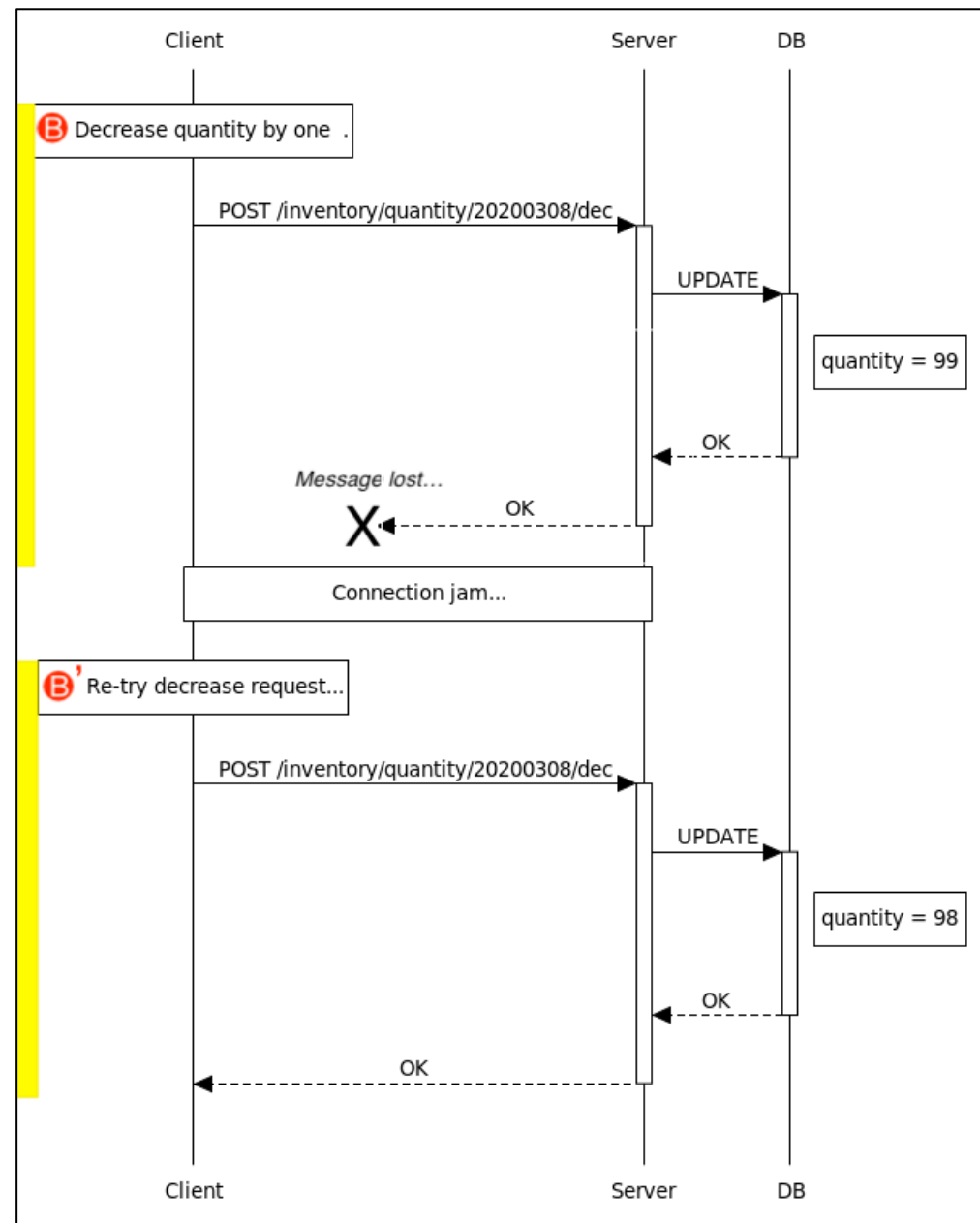
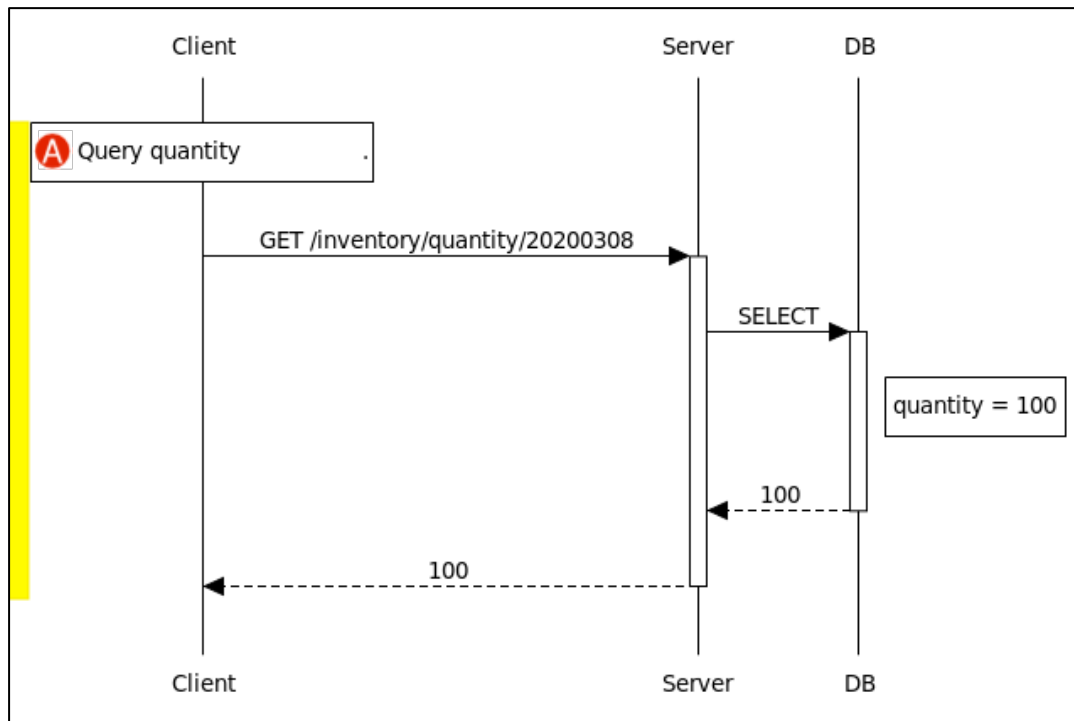
Reconcile loop

```
type MyReconciler struct{}  
⚡  
func (a *MyReconciler) Reconcile(  
    ctx context.Context,  
    req reconcile.Request,  
) (reconcile.Result, error) {  
    fmt.Printf("reconcile %v\n", req)  
    return reconcile.Result{}, nil  
}
```

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/>

部署Operator

Operator 需要那些權限

- Resource
 - MyResource、ConfigMap、Job
- Verbs
 - Create、delete、get、list、patch、update、watch
- Scope
 - 整個Cluster

```
mgr, err := manager.New(  
    config.GetConfigOrDie(),  
    manager.Options{  
        Scheme: scheme,  
    },  
)  
  
if err != nil {  
    panic(err)  
}  
  
err = builder.  
    ControllerManagedBy(mgr).  
    For(&myresourcev1alpha1.MyResource{}).  
    Owns(&corev1.ConfigMap{}).  
    Owns(&batchv1.Job{}).  
    Complete(&MyReconciler{})
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: operator-role
rules:
  - apiGroups:
      - batch
    resources:
      - jobs
    verbs: ""
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs: ""
  - apiGroups:
      - mygroup.example.com
    resources:
      - myresources
    verbs:
      - create
      - delete
      - get
      - list
      - patch
      - update
      - watch
```

```
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
```