极客时间 训练营

模块八： Operator 入门与实战

王炜/ 前腾讯云 CODING 高级架构师

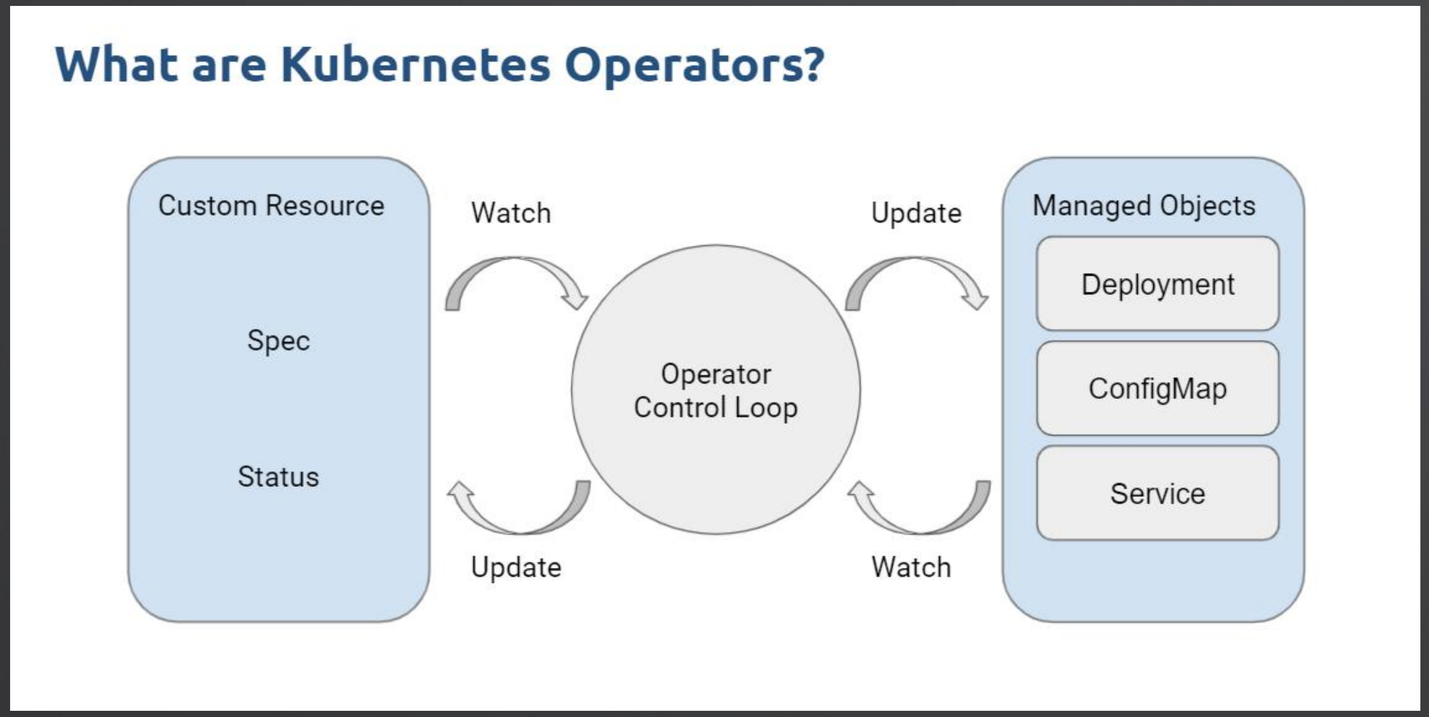
目录

1. Operator 概述

什么是 Operator

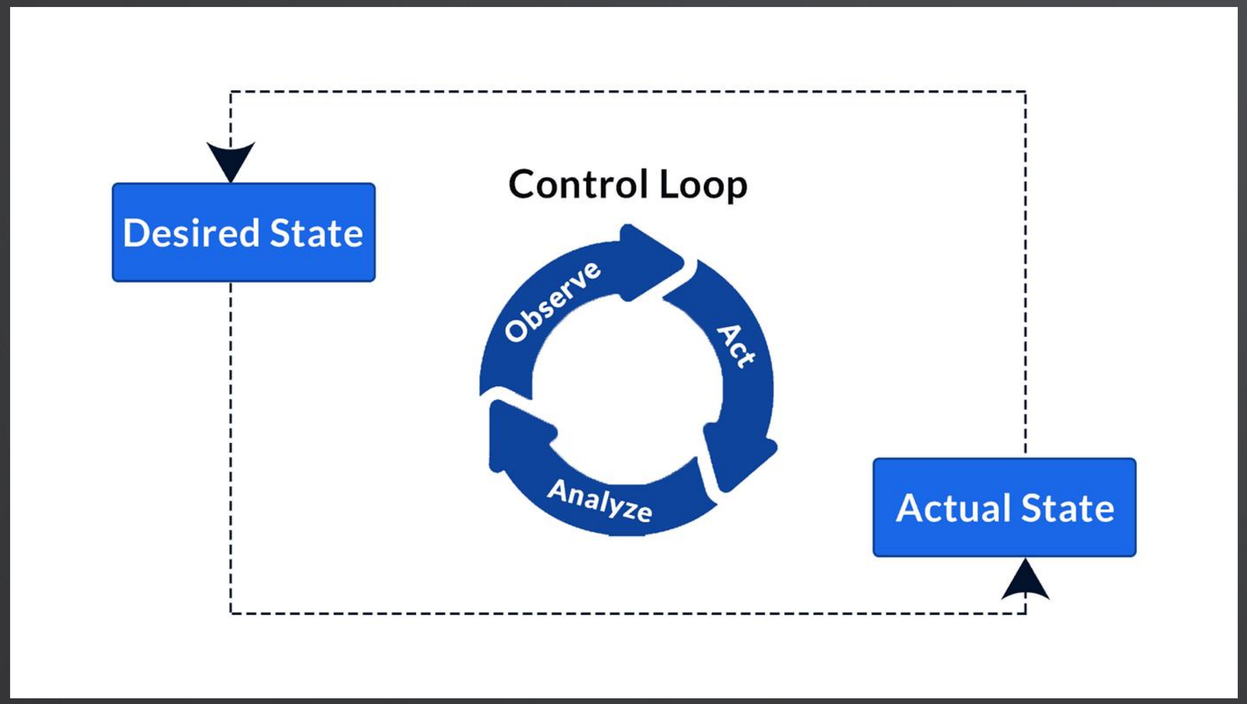
Operator 是一种特殊的控制器（Controller），他能够把控制循环机制应用到自定义资源（CRD）的状态管理中。

Operator = Controller + CRD



期望状态 VS 实际状态

- Spec 字段：期望状态
- Status 字段：实际状态
- Controller：借助控制循环，让期望状态和实际状态保持一致（仅用于 K8s 原生资源：Deployment、Statefulset 等）



replicas.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3  # 期望的副本数
```

replicas.yml

```yaml
status:
  replicas: 3               # 当前实际存在的副本数
  updatedReplicas: 3        # 已更新的副本数
  readyReplicas: 2          # 当前准备就绪的副本数
  availableReplicas: 2      # 可用的副本数
```

CRD

- 自定义资源，借助 CustomResourceDefinition 来描述
- 声明式扩展方式
- 例如：定义一个定时伸缩 HPA CRD 资源（CronHPA）

CRD.yml

```yaml
kind: CustomResourceDefinition
metadata:
  name: cronhpas.example.com  # 自定义资源的名字
spec:
  group: example.com          # API 组
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                crontab:
                  type: string
                  description: # "Cron 表达式，用于定义扩缩容的时间调度。"
                workloadName:
                  type: string
                  description: # "关联工作负载的名字。"
                workloadType:
                  type: string
                  description: # "关联工作负载的类型，比如 Deployment"
```

Operator 开发模式的好处

- 复用 Controller 的控制循环，无需自己编码实现
- 复用 K8s 声明式的资源管理能力
- 和 K8s API 原生集成，并能够使用 kubectl 对资源进行增删改查
- 简化了有状态应用的开发和管理流程
- 能够继承 K8s 的应用管理能力（自愈、滚动更新、自动重启等）

Operator 的使用场景

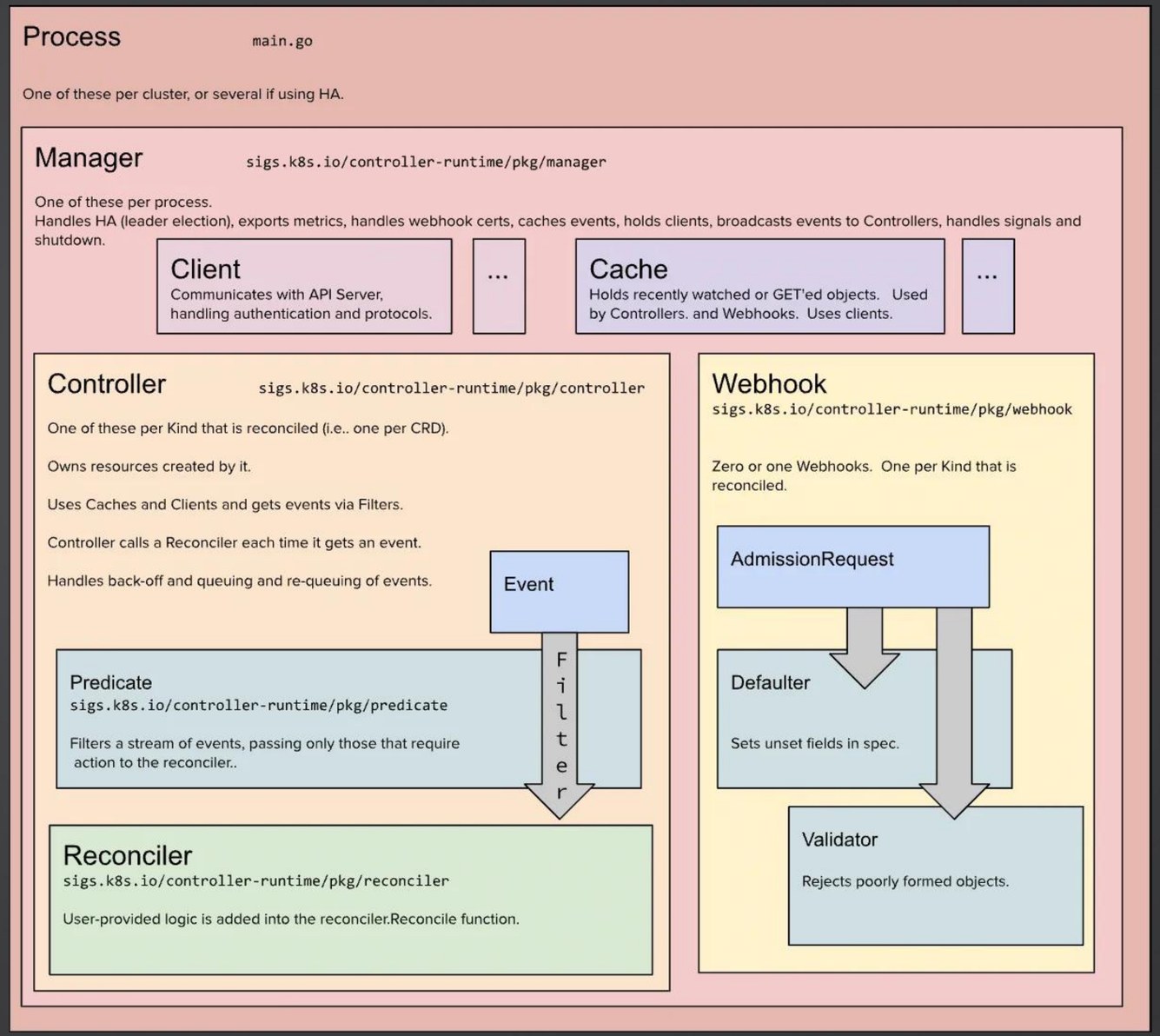
- 自定义资源，例如定义数据库实例、云资源等
- 自动化运维任务，如备份数据库
- CI/CD Workflow
- 存储系统管理：Rook、Ceph 等

Kubebuilder VS Operator SDK

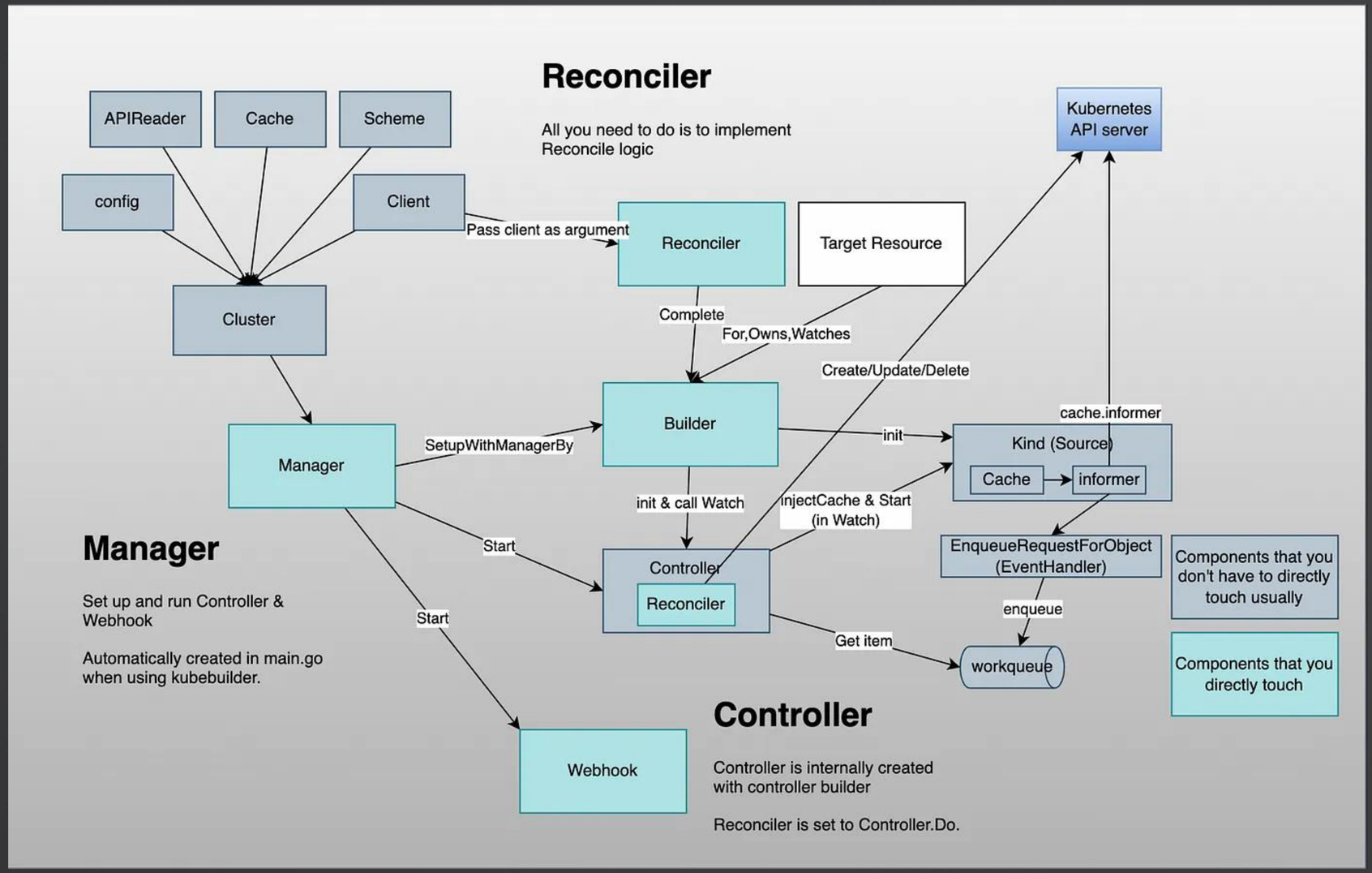
- Kubebuilder：K8s 官方提供的 Operator 开发框架，底层使用了 controller-runtime 和 controller-tools
- Operator SDK：底层使用了 Kubebuilder，但提供了额外的一些能力
  - Operator Lifecycle Manager：很容易打包和分发 Operator
  - OperatorHub：Operator 发布中心，类似 Docker Hub
  - Operator SDK scorecard：小工具，确保 Operator 开发过程的最佳实践
  - 除了使用 Golang 开发，还支持从 Ansible 和 Helm 创建 Operator

Kubebuilder 架构

- Manager：初始化 Controller
- Controller：具备 Cache、队列和失败重试能力
- Reconciler：只需实现这部分业务逻辑
- Client：不直接使用
- Cache：不直接使用
- Webhook：编写 AdmissionWebHook 使用

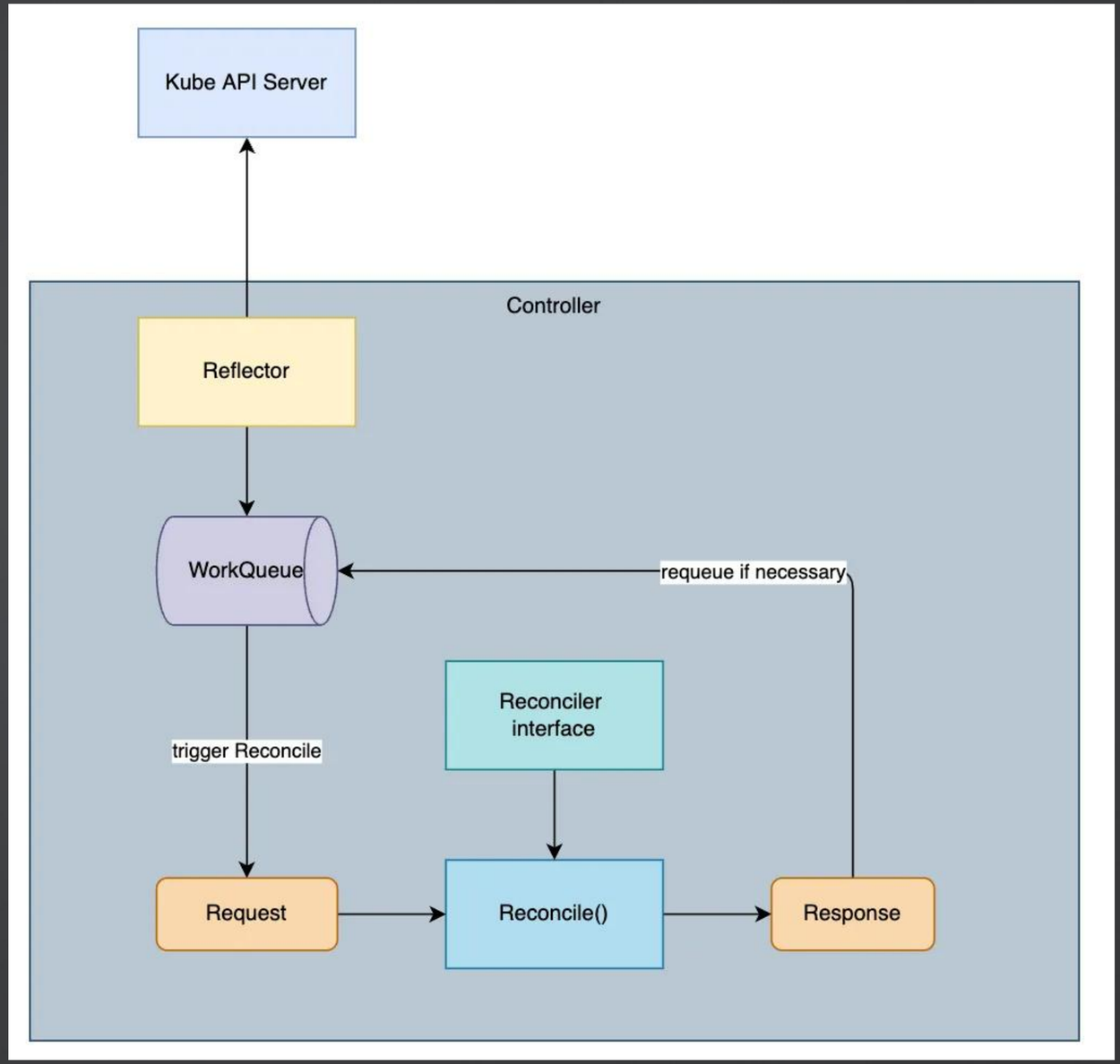


Kubebuilder 架构



Reconcile 架构

- 触发 Reconcile 的过程实际上是从工作队列取获取元素的过程
- 根据 Reconcile 的业务逻辑执行结果，分几种情况
  - 执行成功，无需重试
  - 执行错误，需要重试
  - 执行成功，因其他原因需要重试
  - 其他原因需要等待一段时间后重试



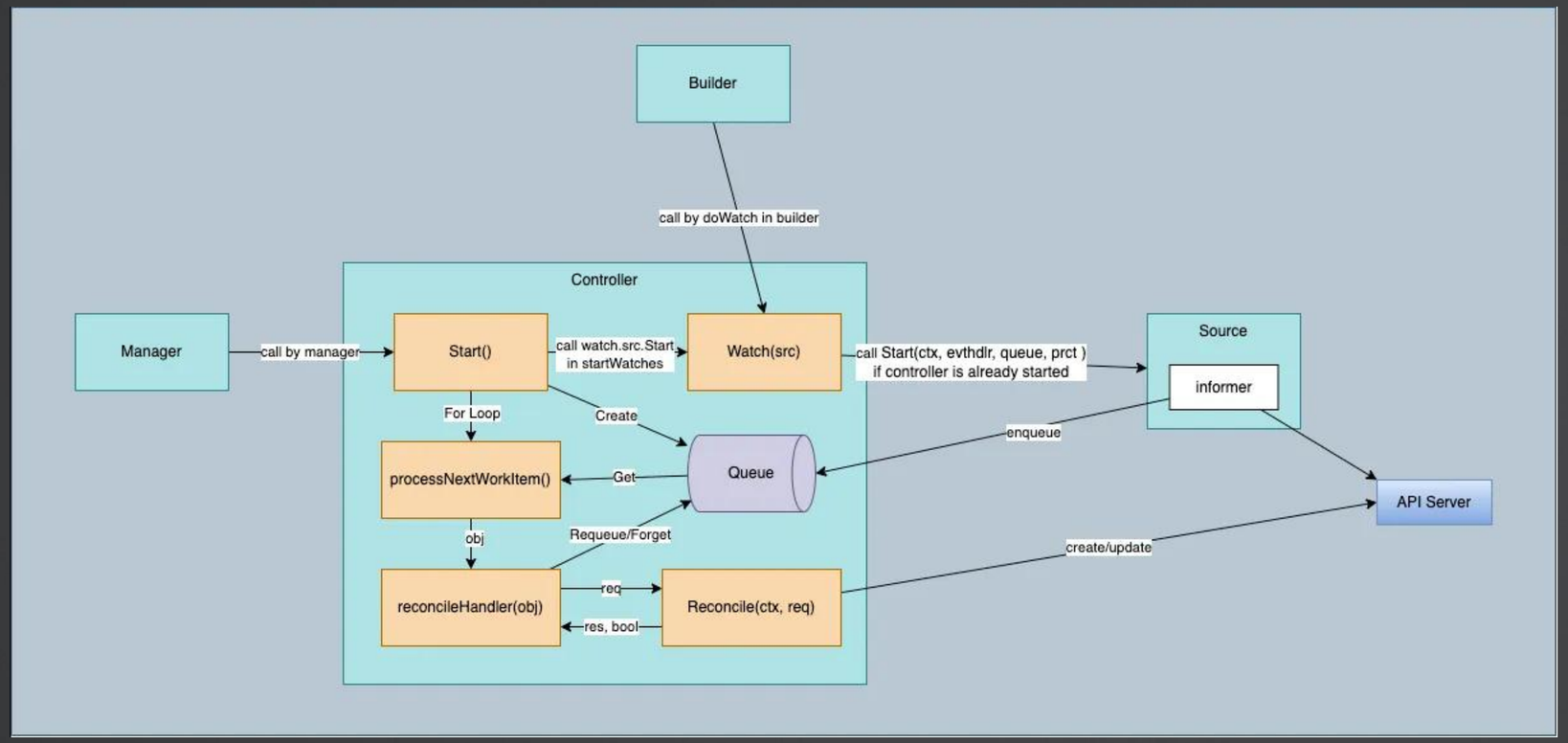
Reconcile 重试

Reconcile.go

```go
func  (r *MyReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error ) {
  ...
  err := someFunc()
   if err != nil {
     return Result{}, err // 返回错误以触发重试
  }
}
```

- 成功，无需重试：return ctrl.Result{}, nil
- 失败，需要重试：return ctrl.Result{}, err
- 成功，其他原因需要重试：return ctrl.Result{Requeue: true}, nil
- 其他原因需要等待一段时间后重试：return ctrl.Result{RequeueAfter: 5 * time.Second}, nil

Controller 架构



- ProcessNextWorkImen()：从工作队列里获取下一个元素
- ReconcileHandler()：调用 Reconcile 方法，并根据返回结果控制元素从队列里删除（业务执行成功），或者重新入队（业务执行失败）

Controller 核心源码

Controller.go

```
1  &controller.Controller{
2    Do: options.Reconciler,
3    MakeQueue: func() workqueue.RateLimitingInterface {
4      return workqueue.NewNamedRateLimitingQueue(options.RateLimiter, name)
5    },
6    MaxConcurrentReconciles: options.MaxConcurrentReconciles,
7    CacheSyncTimeout:        options.CacheSyncTimeout,
8    SetFields:               mgr.SetFields,
9    Name:                    name,
10   LogConstructor:          options.LogConstructor,
11   RecoverPanic:            options.RecoverPanic,
12 }
```

- 使用 RateLimit Queue 来实现工作队列，类似于之前我们在 Client-go 里实现工作队列

2. Kubebuilder 实战一：实现类似 KubeVela Application 定义

本地创建集群

- 使用 Kind 创建本地集群
- https://kind.sigs.k8s.io/docs/user/quick-start/
  - kind create cluster
- 安装 kubebuilder：https://book.kubebuilder.io/quick-start#installation

CRD 设计

- 实现使用一个 CRD 生成 Deployment、Service、Ingress 对象
- Group: application.aiops.com
- Version: v1
- Kind: Application

application.yml

```yaml
apiVersion: application.aiops.com/v1
kind: Application
metadata:
  labels:
    app.kubernetes.io/name: application
    app.kubernetes.io/managed-by: kustomize
  name: application-sample
spec:
  deployment:
    image: nginx
    replicas: 1
    port: 80
  service:
    ports:
      - port: 80
        targetPort: 80
  ingress:
    ingressClassName: nginx
    rules:
      - host: example.foo.com
        http:
          paths:
            - path: /
              pathType: Prefix
              backend:
                service:
                  name: application-sample
                  port:
                    number: 80
```

步骤

- mkdir application && cd application
- go mod init github.com/lyzhang1999/application
- kubebuilder init --domain=aiops.com
- kubebuilder create api --group application --version v1 --kind Application
- 完善 cronhpa/api/v1/application_types.go
  - 修改 ApplicationSpec
- 生成 CRD：make manifests，查看 config/crd/bases/application.aiops.com_applications.yaml 文件
- 编写 internal/controller/application_controller.go Reconcile 业务逻辑
- 将 CRD 安装到集群：make install
- 运行 Operator：make run
- 部署示例：kubectl apply -f config/samples/application_v1_application.yaml

效果

```
> kubectl apply -f config/samples/application_v1_application.yaml
application.application.aiops.com/application-sample created
> kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
application-sample   1/1     1            1           2m2s
nginx                3/3     3            3           132m
> kubectl get application
NAME                 AGE
application-sample   2m10s
> kubectl get svc
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
application-sample   ClusterIP   10.96.142.205   <none>        80/TCP    2m14s
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP   12d
> kubectl get ingress
NAME                 CLASS   HOSTS             ADDRESS   PORTS   AGE
application-sample   nginx   example.foo.com             80      2m18s
```

- 创建 Application 对象，自动创建 Deployment、Service、Ingress 对象
- 删除 Application 对象，自动删除 Deployment、Service、Ingress 对象
  - 这是通过设置 OwnerReference 自动实现的，Reconcile 内并没有写删除资源逻辑

Kubebuilder Project Layout

- /cmd：Operator 主程序入口，当执行 kubebuilder init 时生成
- /api：API 资源定义，用户需要修改 *_types.go 文件来实现自定义 CRD，其他文件为自动生成。当执行 kubebuilder create api 时生成
- /internal/controller：控制器，用户需要修改 *_controller.go 文件来实现自定义 Reconcile 逻辑，当执行 kubebuilder create api 时生成
- /config：CRD 相关对象，自动生成
  - /config/crd
  - /config/rbac
  - /config/sample
- Makefile：自动生成，包含了构建、测试、运行和部署控制器

3. Kubebuilder 实战二：实现阿里云定时弹性伸缩器

CRD 设计

- 实现定时伸缩
- Group: autoscaling.aiops.com
- Version: v1
- Kind: CronHPA

crd.yml

```yaml
apiVersion: autoscaling.aiops.com/v1
kind: CronHPA
metadata:
  name: cronhpa-sample
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  jobs:
    - name: "scale-up"
      schedule: "*/1 * * * *"
      targetSize: 3

```

步骤

- mkdir cronhpa && cd cronhpa
- go mod init github.com/lyzhang1999/cronhpa
- kubebuilder init --domain=aiops.com
- kubebuilder create api --group autoscaling --version v1 --kind CronHPA
- 完善 module_8/cronhpa/api/v1/cronhpa_types.go
  - 增加结构体
  - CronHPA Struct 增加 printcolumn 注释
- 生成 CRD：make manifests，查看 config/crd/bases/autoscaling.aiops.com_cronhpas.yaml 文件
- 编写 internal/controller/cronhpa_controller.go Reconcile 业务逻辑
- 将 CRD 安装到集群：make install
- 运行 Operator：make run
- 部署示例：kubectl apply -f config/samples/autoscaling_v1_cronhpa.yaml

如何打包和部署 Operator

- 设置环境变量：export IMG=lyzhang1999/cronhpa-operator:v0.0.1
- 构建和推送：make docker-build docker-push
- 生成 Operator Deploy 文件：make build-installer
  - 使用 dist/install.yaml 即可在任何的 K8s 集群安装

4. Operator SDK 实战一：基于 Helm 开发 Operator

安装 Operator SDK CLI

- https://sdk.operatorframework.io/docs/installation/

CRD 设计

- 引用 Helm Chart values.yaml 的内容作为 CRD Spec 配置内容
- CRD 出现变化时，自动处理变更（使用新的安装参数重新安装）

redis.yml

```yaml
apiVersion: app.aiops.com/v1
kind: Redis
metadata:
  name: redis-sample
spec:
  # Default values copied from <project_dir>/helm-charts/redis/values.yaml
  architecture: replication
  auth:
    enabled: true
    existingSecret: ""
    existingSecretPasswordKey: ""
    password: ""
    sentinel: true
    usePasswordFileFromSecret: true
    usePasswordFiles: false
```

步骤

- mkdir redis-operator && cd redis-operator
- go mod init github.com/lyzhang1999/redis-operator
- operator-sdk init --domain aiops.com --plugins helm
- operator-sdk create api --group app --version v1 --kind Redis --helm-chart-repo https://charts.bitnami.com/bitnami --helm-chart redis
- 报错，更新 helm 依赖：cd helm-charts/redis && helm dependencies build
- 返回根目录，重新创建 operator-sdk create api --group app --version v1 --kind Redis --helm-chart ./helm-charts/redis
- 构建并推送镜像：make docker-build docker-push IMG="lyzhang1999/redis-operator:v0.0.1"
- 部署 Operator：make deploy IMG="lyzhang1999/redis-operator:v0.0.1"
- 部署 CRD：kubectl apply -f config/samples/app_v1_redis.yaml
- 报错，解决 Service account 权限问题：kubectl create clusterrolebinding redis-operator-cluster-admin \
- --serviceaccount=redis-operator-system:redis-operator-controller-manager \
- --clusterrole=cluster-admin
- 删除 Operator Pod，重新拉起后可见 Redis

5. Operator SDK OLM

Operator Lifecycle Manager (OLM)

- OLM 是用于管理 Operator 生命周期的工具
- 可以实现将 Operator 整体打包，并生成软件部署清单（bundles）
- 很容易安装、卸载、升级 Operator
- 安装：operator-sdk olm install

OLM 使用

- 设置环境变量
  - export IMG=docker.io/lyzhang1999/redis-operator:v0.0.1 // 存储镜像的位置
  - export BUNDLE_IMG=docker.io/lyzhang1999/redis-operator-bundle:v0.0.1 // 存储软件包的位置
- 创建软件包：make bundle
  - 多平台构建可以用 docker buildx build --push --platform linux/amd64 --tag $IMG .
- 推送软件包：make bundle-build bundle-push
- 验证软件包：operator-sdk bundle validate $BUNDLE_IMG
- 安装 Operator（软件包的方式）：operator-sdk run bundle $BUNDLE_IMG

6. 最佳实践

最佳实践

- Reconcile 不应该关注具体事件（创建、更新、删除），更不应该针对不同事件使用不同逻辑
  - 设计一个幂等的 Reconcile 是第一要素（事件无关）
- 幂等的 Reconcile
  - 无论运行多少次，结果都是一样的，因为事件可能会被重复触发
- 简化 Reconcile 逻辑
  - 只关注期望状态和当前状态 Diff，执行业务逻辑

如何对 Operator 进行端到端测试(E2E)?

- 借助 envtest.Environment Mock 一个 K8s API Server
- 安装 setup-event 配置 envtest：go install sigs.k8s.io/controller-runtime/tools/setup-envtest@latest
- 运行：setup-envtest use 1.28
  - 将输出 envtest 二进制文件的目录，例如
    - /Users/wangwei/Library/Application Support/io.kubebuilder.envtest/k8s/1.28.3-darwin-arm64
- 创建软连接
  - sudo mkdir /usr/local/kubebuilder
  - sudo ln -s /path/to/kubebuilder-envtest/k8s/1.23.5-linux-amd64 /usr/local/kubebuilder/bin
- 编写 E2E 测试文件：module_8/application/test/e2e/cluster.go
- 启动测试：make test-e2e

测试结果

```
Ran 1 of 1 Specs in 5.403 seconds
SUCCESS! -- 1 Passed | 0 Failed | 0 Pending | 0 Skipped
You're using deprecated Ginkgo functionality:
============================================
  You are passing a Done channel to a test node to test asynchronous behavior.  This is
synchronously and the timeout will be ignored.
  Learn more at: https://onsi.github.io/ginkgo/MIGRATING_TO_V2#removed-async-testing
    /Users/wangwei/Downloads/aiops/module_8/application/test/e2e/cluster.go:38

To silence deprecations that can be silenced set the following environment variable:
  ACK_GINKGO_DEPRECATIONS=2.14.0

--- PASS: TestE2E (5.40s)
PASS
ok      github.com/lyzhang1999/application/test/e2e     5.873s
```

- 注意：envtest K8s 环境没有真实的控制器，不会有真实的工作负载
- 仅用来测试 Operator CRD 的创建、Status 更新等逻辑

课后作业

- 尝试增强实战二，增加 configmap 字段，实现一并生成 ConfigMap

THANKS