CronJob 开发

AUTHOR: 彭玲 TIME: 2021/11/18

CronJob 开发
- 项目创建
- 控制器开发
  - cronjob_controller.go
    - Reconcile()
    - SetupWithManager()
- webhook 实现
  - main.go
  - cronjob_webhook.go
    - `webhook.Defaulter` 接口实现
    - `webhook.Validator` 接口实现
- 运行 Controller
  - 部署 CRD
  - 本地运行 Controller
  - 部署 CR
  - 资源查看
    - 查看 cronjob
    - 查看 jobs
- cert-manager (证书管理器)
  - 安装
  - 部署文件
- webhooks 部署与测试
  - 构建镜像
  - 修改 kustomize 配置文件
  - 集群部署 webhook
  - 测试 webhook
- Troubleshooting
  - 项目依赖包下载失败
  - gcr.io 镜像下载失败

项目创建

执行下面命令初始化 CronJob 项目。

```
$ kubebuilder init --domain tutorial.kubebuilder.io --repo
tutorial.kb.io/cronjob
$ kubebuilder create api --group batch --version v1 --kind CronJob
```

控制器开发

控制器的工作是确保对于任何给定的对象，实际状态（包括集群状态，以及潜在的外部状态，如 Kubelet 的运行容器或云提供商的负载均衡器）与对象中的期望状态相匹配。

cronjob_controller.go

Reconcile()

```go
// old stuff

import (
    "context"
    "fmt"
    corev1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "sort"
    "time"

    "github.com/go-logr/logr"
    "github.com/robfig/cron"
    kbatch "k8s.io/api/batch/v1"
    "k8s.io/apimachinery/pkg/runtime"
    ref "k8s.io/client-go/tools/reference"
    ctrl "sigs.k8s.io/controller-runtime"
    "sigs.k8s.io/controller-runtime/pkg/client"

    batch "tutorial.kb.io/cronjob/api/v1"
)

func (r *CronJobReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
    // your logic here
    // 1: Load the CronJob by name
    log := r.Log.WithValues("cronjob", req.NamespacedName)

    var cronJob batch.CronJob
    if err := r.Get(ctx, req.NamespacedName, &cronJob); err != nil {
        log.Error(err, "unable to fetch CronJob")
        return ctrl.Result{}, client.IgnoreNotFound(err)
    }

    // 2: List all active jobs, and update the status
    var childJobs kbatch.JobList
    if err := r.List(ctx, &childJobs, client.InNamespace(req.Namespace), client.MatchingFields{jobOwnerKey: req.Name}); err != nil {
        log.Error(err, "unable to list child Jobs")
        return ctrl.Result{}, err
    }

    //  找出所有有效的 job
    var activeJobs []*kbatch.Job
    var successfulJobs []*kbatch.Job
    var failedJobs []*kbatch.Job
    var mostRecentTime *time.Time // 记录其最近一次运行时间以便更新状态

    isJobFinished := func(job *kbatch.Job) (bool, kbatch.JobConditionType) {
        for _, c := range job.Status.Conditions {
```

```
            if (c.Type == kbatch.JobComplete || c.Type == kbatch.JobFailed) && c.Status == corev1.ConditionTrue {
                return true, c.Type
            }
        }

        return false, ""
    }

    var scheduledTimeAnnotation = "batch.tutorial.kubebuilder.io/scheduled-at"

    getScheduledTimeForJob := func(job *kbatch.Job) (*time.Time, error) {
        timeRaw := job.Annotations[scheduledTimeAnnotation]
        if len(timeRaw) == 0 {
            return nil, nil
        }

        timeParsed, err := time.Parse(time.RFC3339, timeRaw)
        if err != nil {
            return nil, err
        }
        return &timeParsed, nil
    }

    for i, job := range childJobs.Items {
        _, finishedType := isJobFinished(&job)
        switch finishedType {
        case "": // ongoing
            activeJobs = append(activeJobs, &childJobs.Items[i])
        case kbatch.JobFailed:
            failedJobs = append(failedJobs, &childJobs.Items[i])
        case kbatch.JobComplete:
            successfulJobs = append(successfulJobs, &childJobs.Items[i])
        }

        // 将启动时间存放在注释中，当job生效时可以从中读取
        scheduledTimeForJob, err := getScheduledTimeForJob(&job)
        if err != nil {
            log.Error(err, "unable to parse schedule time for child job", "job", &job)
            continue
        }
        if scheduledTimeForJob != nil {
            if mostRecentTime == nil {
                mostRecentTime = scheduledTimeForJob
            } else if mostRecentTime.Before(*scheduledTimeForJob) {
                mostRecentTime = scheduledTimeForJob
            }
        }
    }

    if mostRecentTime != nil {
        cronJob.Status.LastScheduleTime = &metav1.Time{Time: *mostRecentTime}
    } else {
        cronJob.Status.LastScheduleTime = nil
    }
```

```
102     cronJob.Status.Active = nil
103     for _, activeJob := range activeJobs {
104         jobRef, err := ref.GetReference(r.Scheme, activeJob)
105         if err != nil {
106             log.Error(err, "unable to make reference to active job", "job",
    activeJob)
107             continue
108         }
109         cronJob.Status.Active = append(cronJob.Status.Active, *jobRef)
110     }
111
112     // 3: Clean up old jobs according to the history limit
113     // NB: deleting these is "best effort" -- if we fail on a particular
    one,
114     // we won't requeue just to finish the deleting.
115     if cronJob.Spec.FailedJobsHistoryLimit != nil {
116         sort.Slice(failedJobs, func(i, j int) bool {
117             if failedJobs[i].Status.StartTime == nil {
118                 return failedJobs[j].Status.StartTime != nil
119             }
120             return
    failedJobs[i].Status.StartTime.Before(failedJobs[j].Status.StartTime)
121         })
122         for i, job := range failedJobs {
123             if int32(i) >= int32(len(failedJobs))-
    *cronJob.Spec.FailedJobsHistoryLimit {
124                 break
125             }
126             if err := r.Delete(ctx, job,
    client.PropagationPolicy(metav1.DeletePropagationBackground));
    client.IgnoreNotFound(err) != nil {
127                 log.Error(err, "unable to delete old failed job", "job",
    job)
128             } else {
129                 log.V(0).Info("deleted old failed job", "job", job)
130             }
131         }
132     }
133
134     if cronJob.Spec.SuccessfulJobsHistoryLimit != nil {
135         sort.Slice(successfulJobs, func(i, j int) bool {
136             if successfulJobs[i].Status.StartTime == nil {
137                 return successfulJobs[j].Status.StartTime != nil
138             }
139             return
    successfulJobs[i].Status.StartTime.Before(successfulJobs[j].Status.StartTim
    e)
140         })
141         for i, job := range successfulJobs {
142             if int32(i) >= int32(len(successfulJobs))-
    *cronJob.Spec.SuccessfulJobsHistoryLimit {
143                 break
144             }
145             if err := r.Delete(ctx, job,
    client.PropagationPolicy(metav1.DeletePropagationBackground)); (err) != nil
    {
146                 log.Error(err, "unable to delete old successful job",
    "job", job)
```

```
147             } else {
148                 log.V(0).Info("deleted old successful job", "job", job)
149             }
150         }
151     }
152
153     // 4: Check if we're suspended
154     if cronJob.Spec.Suspend != nil && *cronJob.Spec.Suspend {
155         log.V(1).Info("cronjob suspended, skipping")
156         return ctrl.Result{}, nil
157     }
158
159     // 5: Get the next scheduled run
160     getNextSchedule := func(cronJob *batch.CronJob, now time.Time)
(lastMissed time.Time, next time.Time, err error) {
161         sched, err := cron.ParseStandard(cronJob.Spec.Schedule)
162         if err != nil {
163             return time.Time{}, time.Time{}, fmt.Errorf("Unparseable
schedule %q: %v", cronJob.Spec.Schedule, err)
164         }
165
166         // for optimization purposes, cheat a bit and start from our last
observed run time
167         // we could reconstitute this here, but there's not much point,
since we've
168         // just updated it.
169         var earliestTime time.Time
170         if cronJob.Status.LastScheduleTime != nil {
171             earliestTime = cronJob.Status.LastScheduleTime.Time
172         } else {
173             earliestTime = cronJob.ObjectMeta.CreationTimestamp.Time
174         }
175         if cronJob.Spec.StartingDeadlineSeconds != nil {
176             // controller is not going to schedule anything below this
point
177             schedulingDeadline := now.Add(-time.Second *
time.Duration(*cronJob.Spec.StartingDeadlineSeconds))
178
179             if schedulingDeadline.After(earliestTime) {
180                 earliestTime = schedulingDeadline
181             }
182         }
183         if earliestTime.After(now) {
184             return time.Time{}, sched.Next(now), nil
185         }
186
187         starts := 0
188         for t := sched.Next(earliestTime); !t.After(now); t = sched.Next(t)
{
189             lastMissed = t
190             // An object might miss several starts. For example, if
191             // controller gets wedged on Friday at 5:01pm when everyone has
192             // gone home, and someone comes in on Tuesday AM and discovers
193             // the problem and restarts the controller, then all the hourly
194             // jobs, more than 80 of them for one hourly scheduledJob,
should
195             // all start running with no further intervention (if the
scheduledJob
```

```
            // allows concurrency and late starts).
            //
            // However, if there is a bug somewhere, or incorrect clock
            // on controller's server or apiservers (for setting creationTimestamp)
            // then there could be so many missed start times (it could be off
            // by decades or more), that it would eat up all the CPU and memory
            // of this controller. In that case, we want to not try to list
            // all the missed start times.
            starts++
            if starts > 100 {
                // We can't get the most recent times so just return an empty slice
                return time.Time{}, time.Time{}, fmt.Errorf("Too many missed start times (> 100). Set or decrease .spec.startingDeadlineSeconds or check clock skew.")
            }
        }
        return lastMissed, sched.Next(now), nil
    }
    // 计算出定时任务下一次执行时间（或是遗漏的执行时间）
    missedRun, nextRun, err := getNextSchedule(&cronJob, r.Now())
    if err != nil {
        log.Error(err, "unable to figure out CronJob schedule")
        // 重新排队直到有更新修复这次定时任务调度，不必返回错误
        return ctrl.Result{}, nil
    }

    scheduledResult := ctrl.Result{RequeueAfter: nextRun.Sub(r.Now())} // 保存以便别处复用
    log = log.WithValues("now", r.Now(), "next run", nextRun)

    // 6: Run a new job if it’s on schedule, not past the deadline, and not blocked by our concurrency policy
    if missedRun.IsZero() {
        log.V(1).Info("no upcoming scheduled times, sleeping until next")
        return scheduledResult, nil
    }

    // 确保错过的执行没有超过截止时间
    log = log.WithValues("current run", missedRun)
    tooLate := false
    if cronJob.Spec.StartingDeadlineSeconds != nil {
        tooLate = missedRun.Add(time.Duration(*cronJob.Spec.StartingDeadlineSeconds) * time.Second).Before(r.Now())
    }
    if tooLate {
        log.V(1).Info("missed starting deadline for last run, sleeping till next")
        // TODO(directxman12): events
        return scheduledResult, nil
    }

    // figure out how to run this job -- concurrency policy might forbid us from running
```

```
242     // multiple at the same time...
243     if cronJob.Spec.ConcurrencyPolicy == batch.ForbidConcurrent &&
    len(activeJobs) > 0 {
244         log.V(1).Info("concurrency policy blocks concurrent runs,
    skipping", "num active", len(activeJobs))
245         return scheduledResult, nil
246     }
247
248     // ...or instruct us to replace existing ones...
249     if cronJob.Spec.ConcurrencyPolicy == batch.ReplaceConcurrent {
250         for _, activeJob := range activeJobs {
251             // we don't care if the job was already deleted
252             if err := r.Delete(ctx, activeJob,
    client.PropagationPolicy(metav1.DeletePropagationBackground));
    client.IgnoreNotFound(err) != nil {
253                 log.Error(err, "unable to delete active job", "job",
    activeJob)
254                 return ctrl.Result{}, err
255             }
256         }
257     }
258
259     constructJobForCronJob := func(cronJob *batch.CronJob, scheduledTime
    time.Time) (*kbatch.Job, error) {
260         // job 名称带上执行时间以确保唯一性，避免排定执行时间的 job 创建两次
261         name := fmt.Sprintf("%s-%d", cronJob.Name, scheduledTime.Unix())
262
263         job := &kbatch.Job{
264             ObjectMeta: metav1.ObjectMeta{
265                 Labels:      make(map[string]string),
266                 Annotations: make(map[string]string),
267                 Name:        name,
268                 Namespace:   cronJob.Namespace,
269             },
270             Spec: *cronJob.Spec.JobTemplate.Spec.DeepCopy(),
271         }
272         for k, v := range cronJob.Spec.JobTemplate.Annotations {
273             job.Annotations[k] = v
274         }
275         job.Annotations[scheduledTimeAnnotation] =
    scheduledTime.Format(time.RFC3339)
276         for k, v := range cronJob.Spec.JobTemplate.Labels {
277             job.Labels[k] = v
278         }
279         if err := ctrl.SetControllerReference(cronJob, job, r.Scheme); err
    != nil {
280             return nil, err
281         }
282
283         return job, nil
284     }
285
286     // actually make the job...
287     job, err := constructJobForCronJob(&cronJob, missedRun)
288     if err != nil {
289         log.Error(err, "unable to construct job from template")
290         // don't bother requeuing until we get a change to the spec
291         return scheduledResult, nil
```

```
    }

    // ...and create it on the cluster
    if err := r.Create(ctx, job); err != nil {
        log.Error(err, "unable to create Job for CronJob", "job", job)
        return ctrl.Result{}, err
    }

    log.V(1).Info("created Job for CronJob run", "job", job)

    // 7: Requeue when we either see a running job or it's time for the next scheduled run
    // we'll requeue once we see the running job, and update our status
    return scheduledResult, nil

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

SetupWithManager()

```
// SetupWithManager sets up the controller with the Manager.
func (r *CronJobReconciler) SetupWithManager(mgr ctrl.Manager) error {
    // 此处不是测试，我们需要创建一个真实的时钟
    if r.Clock == nil {
        r.Clock = realClock{}
    }

    if err := mgr.GetFieldIndexer().IndexField(context.Background(), &kbatch.Job{}, jobOwnerKey, func(rawObj client.Object) []string {
        //获取 job 对象，提取 owner...
        job := rawObj.(*kbatch.Job)
        owner := metav1.GetControllerOf(job)
        if owner == nil {
            return nil
        }
        // ...确保 owner 是个 CronJob...
        if owner.APIVersion != apiGVStr || owner.Kind != "CronJob" {
            return nil
        }

        // ...是 CronJob，返回
        return []string{owner.Name}
    }); err != nil {
        return err
    }

    return ctrl.NewControllerManagedBy(mgr).
        For(&batch.CronJob{}).
        Owns(&kbatch.Job{}).
        Complete(r)
}
```

webhook 实现

运行下面的命令为 CronJob (CRD) 创建一个 webhooks 脚手架。（带上`--defaulting`和`--programmatic-validation`标志，因为测试项目会用到`默认`和`验证`webhooks 。）

```
julin@FSZJ-PENGLING:~/myproject/cronjob$ kubebuilder create webhook --group batch --version v1 --kind CronJob --defaulting --programmatic-validation
Writing kustomize manifests for you to edit...
Writing scaffold for you to edit...
api/v1/cronjob_webhook.go
```

main.go

执行`kubebuilder create webhook`命令后，main.go 增加如下代码：

```
julin@FSZJ-PENGLING:~/myproject$ diff cronjob_create_api/main.go cronjob_create_webhook/main.go
87a88,91
>   if err = (&batchv1.CronJob{}).SetupWebhookWithManager(mgr); err != nil {
>       setupLog.Error(err, "unable to create webhook", "webhook", "CronJob")
>       os.Exit(1)
>   }
```

cronjob_webhook.go

`webhook.Defaulter`接口实现

```go
var _ webhook.Defaulter = &CronJob{}

// Default implements webhook.Defaulter so a webhook will be registered for the type
func (r *CronJob) Default() {
    cronjoblog.Info("default", "name", r.Name)

    // TODO(user): fill in your defaulting logic.
    if r.Spec.ConcurrencyPolicy == "" {
        r.Spec.ConcurrencyPolicy = AllowConcurrent
    }
    if r.Spec.Suspend == nil {
        r.Spec.Suspend = new(bool)
    }
    if r.Spec.SuccessfulJobsHistoryLimit == nil {
        r.Spec.SuccessfulJobsHistoryLimit = new(int32)
        *r.Spec.SuccessfulJobsHistoryLimit = 3
    }
    if r.Spec.FailedJobsHistoryLimit == nil {
        r.Spec.FailedJobsHistoryLimit = new(int32)
        *r.Spec.FailedJobsHistoryLimit = 1
    }
}
```

`webhook.Validator`接口实现

```
var _ webhook.Validator = &CronJob{}

// ValidateCreate implements webhook.Validator so a webhook will be registered for the type
func (r *CronJob) ValidateCreate() error {
    cronjoblog.Info("validate create", "name", r.Name)

    // TODO(user): fill in your validation logic upon object creation.
    return r.validateCronJob()
}

// ValidateUpdate implements webhook.Validator so a webhook will be registered for the type
func (r *CronJob) ValidateUpdate(old runtime.Object) error {
    cronjoblog.Info("validate update", "name", r.Name)

    // TODO(user): fill in your validation logic upon object update.
    return r.validateCronJob()
}

// ValidateDelete implements webhook.Validator so a webhook will be registered for the type
func (r *CronJob) ValidateDelete() error {
    cronjoblog.Info("validate delete", "name", r.Name)

    // TODO(user): fill in your validation logic upon object deletion.
    return nil
}
```

运行 Controller

部署 CRD

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ make install
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/controller-gen "crd:trivialVersions=true,preserveUnknownFields=false" rbac:roleName=manager-role webhook paths="./..." output:crd:artifacts:config=config/crd/bases
go: creating new go.mod: module tmp
Downloading sigs.k8s.io/kustomize/kustomize/v3@v3.8.7
go get: added sigs.k8s.io/kustomize/kustomize/v3 v3.8.7
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/kustomize build config/crd | kubectl apply -f -
customresourcedefinition.apiextensions.k8s.io/cronjobs.batch.tutorial.kubebuilder.io created
```

本地运行 Controller

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ make run ENABLE_WEBHOOKS=false
```

```
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/controller-gen "crd:trivialVersions=true,preserveUnknownFields=false" rbac:roleName=manager-role webhook paths="./..." output:crd:artifacts:config=config/crd/bases
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/controller-gen object:headerFile="hack/boilerplate.go.txt" paths="./..."
go fmt ./...
go vet ./...
go run ./main.go
I1102 16:33:18.716259    7596 request.go:655] Throttling request took 1.018578497s, request: GET:https://10.8.30.38:6443/apis/snapshot.storage.k8s.io/v1beta1?timeout=32s
2021-11-02T16:33:19.320+0800    INFO    controller-runtime.metrics  metrics server is starting to listen    {"addr": ":8080"}
2021-11-02T16:33:19.322+0800    INFO    setup   starting manager
2021-11-02T16:33:19.428+0800    INFO    controller-runtime.manager  starting metrics server {"path": "/metrics"}
2021-11-02T16:33:19.428+0800    INFO    controller-runtime.manager.controller.cronjob   Starting EventSource    {"reconciler group": "batch.tutorial.kubebuilder.io", "reconciler kind": "CronJob", "source": "kind source: /, Kind="}
2021-11-02T16:33:19.529+0800    INFO    controller-runtime.manager.controller.cronjob   Starting EventSource    {"reconciler group": "batch.tutorial.kubebuilder.io", "reconciler kind": "CronJob", "source": "kind source: /, Kind="}
2021-11-02T16:33:19.529+0800    INFO    controller-runtime.manager.controller.cronjob   Starting Controller {"reconciler group": "batch.tutorial.kubebuilder.io", "reconciler kind": "CronJob"}
2021-11-02T16:33:19.529+0800    INFO    controller-runtime.manager.controller.cronjob   Starting workers    {"reconciler group": "batch.tutorial.kubebuilder.io", "reconciler kind": "CronJob", "worker count": 1}
```

结果输出 controller 关于启动的日志，但它还没有做任何事情。

部署 CR

编写一个 CronJob 的配置文件`config/samples/batch_v1_cronjob.yaml`进行测试。

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ vi config/samples/batch_v1_cronjob.yaml

apiVersion: batch.tutorial.kubebuilder.io/v1
kind: CronJob
metadata:
  name: cronjob-sample
spec:
  # Add fields here
  foo: bar
  schedule: "*/1 * * * *"
  startingDeadlineSeconds: 60
  concurrencyPolicy: Allow # explicitly specify, but Allow is also default.
  jobTemplate:
    spec:
      template:
        spec:
          containers:
```

```
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

部署 CronJob 对象：

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ kubectl create -f config/samples/batch_v1_cronjob.yaml
cronjob.batch.tutorial.kubebuilder.io/cronjob-sample created
```

控制器输出：

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ make run ENABLE_WEBHOOKS=false
...
2021-11-02T16:41:06.989+0800    DEBUG   controllers.CronJob no upcoming scheduled times, sleeping until next    {"cronjob": "default/cronjob-sample", "now": "2021-11-02T16:41:06.989+0800", "next run": "2021-11-02T16:42:00.000+0800"}
2021-11-02T16:42:01.626+0800    DEBUG   controllers.CronJob created Job for CronJob run {"cronjob": "default/cronjob-sample", "now": "2021-11-02T16:42:00.001+0800", "next run": "2021-11-02T16:43:00.000+0800", "current run": "2021-11-02T16:42:00.000+0800", "job": {"namespace": "default", "name": "cronjob-sample-1635842520"}}
2021-11-02T16:42:01.626+0800    DEBUG   controllers.CronJob no upcoming scheduled times, sleeping until next    {"cronjob": "default/cronjob-sample", "now": "2021-11-02T16:42:01.626+0800", "next run": "2021-11-02T16:43:00.000+0800"}
2021-11-02T16:42:04.165+0800    DEBUG   controllers.CronJob no upcoming scheduled times, sleeping until next    {"cronjob": "default/cronjob-sample", "now": "2021-11-02T16:42:04.165+0800", "next run": "2021-11-02T16:43:00.000+0800"}

```

资源查看

查看 cronjob

```
anxin@node38:~$ kubectl get cronjob.batch.tutorial.kubebuilder.io cronjob-sample -o yaml
apiVersion: batch.tutorial.kubebuilder.io/v1
kind: CronJob
metadata:
  creationTimestamp: "2021-11-02T08:41:06Z"
  generation: 1
  managedFields:
  - apiVersion: batch.tutorial.kubebuilder.io/v1
    fieldsType: FieldsV1
    fieldsV1:
      f:spec:
        .: {}
```

```
        f:concurrencyPolicy: {}
        f:foo: {}
        f:jobTemplate:
          .: {}
          f:spec:
            .: {}
            f:template:
              .: {}
              f:spec:
                .: {}
                f:containers: {}
                f:restartPolicy: {}
        f:schedule: {}
        f:startingDeadlineSeconds: {}
    manager: kubectl
    operation: Update
    time: "2021-11-02T08:41:06Z"
  name: cronjob-sample # 名称
  namespace: default
  resourceVersion: "141580680"
  selfLink:
/apis/batch.tutorial.kubebuilder.io/v1/namespaces/default/cronjobs/cronjob-
sample
  uid: df27a93b-4c2e-4ff0-9d98-5534c1bcd8f2
spec:
  concurrencyPolicy: Allow
  foo: bar
  jobTemplate: # Job 模板
    spec:
      template:
        spec:
          containers:
          - args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
            image: busybox
            name: hello
          restartPolicy: OnFailure
  schedule: '*/1 * * * *'
  startingDeadlineSeconds: 60
```

查看 jobs

```
anxin@node38:~$ kubectl get job
NAME                     COMPLETIONS      DURATION    AGE
cronjob-sample-1635842520     0/1           27m        27m
cronjob-sample-1635842580     0/1           26m        26m
cronjob-sample-1635842640     0/1           25m        25m
cronjob-sample-1635842700     0/1           24m        24m
cronjob-sample-1635842760     0/1           23m        23m
```

cert-manager (证书管理器)

cert-manager 在 Kubernetes 集群中添加证书和证书颁发者作为资源类型，简化了获取、更新和使用这些证书的过程。

cert-manager 将确保证书是有效的和最新的，并尝试在过期之前的配置时间更新证书。

安装

不需要对 cert-manager `安装参数` 进行任何调整。

```
anxin@node38:~$ kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.6.0/cert-manager.yaml
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
namespace/cert-manager created # 创建 Namespace : cert-manager
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrole.rbac.authorization.k8s.io/cert-manager-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-edit created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrole.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-orders created
```

```
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
role.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
role.rbac.authorization.k8s.io/cert-manager:leaderelection created
role.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
rolebinding.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
service/cert-manager created # 创建 Service : cert-manager (端口 9402)
service/cert-manager-webhook created # 创建 Service : cert-manager-webhook (端口 443)
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created
```

部署文件

Cert manager 有一个叫做 CA 注入器的组件，该组件负责将 CA 捆绑注入到 `Mutating` | `Validating` WebhookConfiguration 中。为此，需要使用带有 key 为 `cert-manager.io/inject-ca-from` 的注释。

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob/config/default$ vi webhookcainjection_patch.yaml

# This patch add annotation to admission webhook config and
# the variables $(CERTIFICATE_NAMESPACE) and $(CERTIFICATE_NAME) will be substituted by kustomize.
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: mutating-webhook-configuration
  annotations:
    cert-manager.io/inject-ca-from: $(CERTIFICATE_NAMESPACE)/$(CERTIFICATE_NAME)
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: validating-webhook-configuration
  annotations:
```

```
    cert-manager.io/inject-ca-from:
$(CERTIFICATE_NAMESPACE)/$(CERTIFICATE_NAME)
```

webhooks 部署与测试

构建镜像

运行`make docker-build`进行本地构建镜像。或者使用`docker build -t cronjob-controller .`直接构建镜像：

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ docker build -t cronjob-
controller .
```

修改 kustomize 配置文件

启用 webhook 和证书管理配置：config/default/kustomization.yaml 和 config/crd/kustomization.yaml 。

集群部署 webhook

通过下面的命令部署到集群中。webhook 的 pod 启动大概需要 1 分钟，并且提供了证书认证。

```
# make deploy IMG=<some-registry>/<project-name>:tag
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ make deploy IMG=cronjob-
controller
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/controller-gen
"crd:trivialVersions=true,preserveUnknownFields=false"
rbac:roleName=manager-role webhook paths="./..."
output:crd:artifacts:config=config/crd/bases
cd config/manager &&
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/kustomize edit set image
controller=cronjob-controller
/home/anxin/pengling/k8s/kubebuilder/cronjob/bin/kustomize build
config/default | kubectl apply -f -
namespace/cronjob-system created

customresourcedefinition.apiextensions.k8s.io/cronjobs.batch.tutorial.kubebu
ilder.io configured
serviceaccount/cronjob-controller-manager created
role.rbac.authorization.k8s.io/cronjob-leader-election-role created
clusterrole.rbac.authorization.k8s.io/cronjob-manager-role created
clusterrole.rbac.authorization.k8s.io/cronjob-metrics-reader created
clusterrole.rbac.authorization.k8s.io/cronjob-proxy-role created
rolebinding.rbac.authorization.k8s.io/cronjob-leader-election-rolebinding
created
clusterrolebinding.rbac.authorization.k8s.io/cronjob-manager-rolebinding
created
clusterrolebinding.rbac.authorization.k8s.io/cronjob-proxy-rolebinding
created
configmap/cronjob-manager-config created
service/cronjob-controller-manager-metrics-service created
service/cronjob-webhook-service created
deployment.apps/cronjob-controller-manager created
```

```
mutatingwebhookconfiguration.admissionregistration.k8s.io/cronjob-mutating-webhook-configuration created
validatingwebhookconfiguration.admissionregistration.k8s.io/cronjob-validating-webhook-configuration created
Error from server (InternalError): error when creating "STDIN": Internal error occurred: failed calling webhook "webhook.cert-manager.io": Post https://cert-manager-webhook.cert-manager.svc:443/mutate?timeout=10s: dial tcp 10.1.177.12:443: connect: connection refused # 无法访问 https 服务，缺少对应证书
Error from server (InternalError): error when creating "STDIN": Internal error occurred: failed calling webhook "webhook.cert-manager.io": Post https://cert-manager-webhook.cert-manager.svc:443/mutate?timeout=10s: dial tcp 10.1.177.12:443: connect: connection refused
Makefile:81: recipe for target 'deploy' failed
make: *** [deploy] Error 1
```

测试 webhook

现在你可以创建一个有效的 CronJob 来测试你的 webhook。

```
kubectl create -f config/samples/batch_v1_cronjob.yaml
```

node38 集群环境问题，导致 https 缺少 TLS 证书：

```
anxin@node38:~/pengling/k8s/kubebuilder/cronjob$ kubectl create -f config/samples/batch_v1_cronjob.yaml
Error from server (InternalError): error when creating "config/samples/batch_v1_cronjob.yaml": Internal error occurred: failed calling webhook "mcronjob.kb.io": Post https://cronjob-webhook-service.cronjob-system.svc:443/mutate-batch-tutorial-kubebuilder-io-v1-cronjob?timeout=10s: dial tcp 10.1.192.162:443: connect: connection refused # 无法访问 https 服务，缺少对应证书
```

Troubleshooting

项目依赖包下载失败

执行 `make docker-build` 至 Dockerfile 第 5 步 `RUN go mod download` 时，从 https://proxy.golang.org 默认代理下载项目依赖，失败。

解决方案：Dockerfile 第 5 步下载依赖包之前，设置 go 国内代理 https://goproxy.cn 。

```
# cache deps before building and copying source so that we don't need to re-download as much
# and so that source changes don't invalidate our downloaded layer
RUN export GOPROXY=https://goproxy.cn,direct && go mod download
```

gcr.io 镜像下载失败

解决方案：利用阿里云构建镜像，使用该镜像。

```
# Use distroless as minimal base image to package the manager binary
# Refer to https://github.com/GoogleContainerTools/distroless for more details
# FROM gcr.io/distroless/static:nonroot
FROM registry.cn-hangzhou.aliyuncs.com/gcr_containerx/distroless:nonroot
```