kubebuilder 介绍

本节主要介绍 Kubernetes 源码获取以及编译环境、IDE 配置等。

介绍

在 Kubernetes 中开发 Operator 的时候，我们肯定需要使用到 CRD 以及对应的 Controller ，我们可以通过 CRD 定义业务相关的资源，并利用 controller 实现对应的业务逻辑，例如创建/删除 deployment，并根据资源变化做出相应的动作。但是如果全都去手动生成代码，然后再来编写业务代码显得有点麻烦，为此在社区中，为我们提供了基于 CRD 开发的框架，主要有 kubebuilder 以及 operator-sdk 两个框架，这两者大同小异，都是利用兴趣小组提供的 `controller-runtime` 项目实现的 Controller 逻辑，不同的是 CRD 资源的创建过程。

本文我们先来简单介绍下 kubebuilder，kubebuilder 由 Kubernetes 特殊兴趣组(SIG) API Machinery 拥有和维护，能够帮助开发者创建 CRD 并生成 controller 脚手架，下面我们来简单使用下 kubebuilder。

1. 安装

```
$ os=$(go env GOOS)
$ arch=$(go env GOARCH)

# 下载 kubebuilder 并解压到 tmp 目录中
$ curl -L https://go.kubebuilder.io/dl/2.3.1/${os}/${arch} | tar -xz -C /tmp/

# 将 kubebuilder 移动 PATH 路径中
$ sudo mv /tmp/kubebuilder_2.3.1_${os}_${arch} /usr/local/bin/kubebuilder
$ kubebuilder version
Version: version.Version{KubeBuilderVersion:"2.3.1", KubernetesVendor:"1.16.4", GitCommit:"8b53abeb4280186e494b726edf8f54ca7aa64a49",
```

2. 新建项目

创建一个目录，然后在里面运行 `kubebuilder init` 命令，初始化一个新项目。示例如下。

```
$ mkdir github.com/cnych/builder-demo
$ cd github.com/cnych/builder-demo
# 开启 go modules
$ export GO111MODULE=on
$ export GOPROXY=https://goproxy.cn
# 初始化项目
$ kubebuilder init --domain ydzs.io --owner cnych --repo github.com/cnych/builder-demo
Writing scaffold for you to edit...
Get controller runtime:
$ go get sigs.k8s.io/controller-runtime@v0.5.0
go: downloading sigs.k8s.io/controller-runtime v0.5.0
go: downloading k8s.io/apimachinery v0.17.2
go: downloading k8s.io/client-go v0.17.2
go: downloading k8s.io/api v0.17.2
go: downloading gomodules.xyz/jsonpatch/v2 v2.0.1
```

```
go: downloading k8s.io/apiextensions-apiserver v0.17.2
go: downloading github.com/googleapis/gnostic v0.3.1
go: downloading k8s.io/kube-openapi v0.0.0-20191107075043-30be4d16710a
go: downloading github.com/golang/groupcache v0.0.0-20180513044358-24b0969c4cb7
go: downloading golang.org/x/crypto v0.0.0-20190820162420-60c769a6c586
go: downloading github.com/imdario/mergo v0.3.6
Update go.mod:
$ go mod tidy
Running make:
$ make
/Users/ych/devs/projects/go/bin/controller-gen object:headerFile="hack/boilerplate.go.txt" paths="./..."
go fmt ./...
go vet ./...
go build -o bin/manager main.go
Next: define a resource with:
$ kubebuilder create api
```

新建一个 API

运行下面的命令，创建一个新的 API（组/版本）为 “webapp/v1”，并在上面创建新的 Kind(CRD) “Guestbook”。

```
$ kubebuilder create api --group webapp --version v1 --kind Guestbook
Create R+esource [y/n]
y
Create Controller [y/n]
y
Writing scaffold for you to edit...
api/v1/guestbook_types.go
controllers/guestbook_controller.go
Running make:
$ make
/Users/ych/devs/projects/go/bin/controller-gen object:headerFile="hack/boilerplate.go.txt" paths="./..."
go fmt ./...
go vet ./...
go build -o bin/manager main.go
```

上面的命令会创建文件 `api/v1/guestbook_types.go` ，该文件中定义相关 API ，而针对于这一类型 (CRD) 的业务逻辑生成在 `controller/guestbook_controller.go` 文件中。

我们可以根据自己的需求去修改资源对象的定义结构，修改 `api/v1/guestbook_types.go` 文件：

```
# api/v1/guestbook_types.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"`
}

// GuestbookStatus defines the observed state of Guestbook
type GuestbookStatus struct {
    // INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
    // Important: Run "make" to regenerate code after modifying this file

    // PodName of the active Guestbook node.
    Active string `json:"active"`

    // PodNames of the standby Guestbook nodes.
    Standby []string `json:"standby"`
}

// +kubebuilder:object:root=true
// +kubebuilder:subresource:status
// +kubebuilder:resource:scope=Cluster

// Guestbook is the Schema for the guestbooks API
type Guestbook struct {
    metav1.TypeMeta   `json:",inline"`
```

```
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   GuestbookSpec   `json:"spec,omitempty"`
    Status GuestbookStatus `json:"status,omitempty"`
}
```

3. 测试

定义完成后我们需要一个 Kubernetes 集群来测试 CRD运行，控制器将自动使用你的 `kubeconfig` 文件中的当前上下文来连接集群，所以首先确保可以正常访问集群。

将 CRD 安装到集群中，执行如下命令即可：

```
$ make install
```

然后使用下面的命令运行控制器：

```
$ make run
/Users/ych/devs/projects/go/bin/controller-gen object:headerFile="hack/boilerplate.go.txt" paths="./..."
go fmt ./...
go vet ./...
/Users/ych/devs/projects/go/bin/controller-gen "crd:trivialVersions=true" rbac:roleName=manager-role webhook paths="./..." output:crd:
go run ./main.go
2020-10-14T18:07:38.849+0800    INFO    controller-runtime.metrics      metrics server is starting to listen    {"addr": ":8080"}
2020-10-14T18:07:38.850+0800    INFO    setup   starting manager
2020-10-14T18:07:38.850+0800    INFO    controller-runtime.manager      starting metrics server {"path": "/metrics"}
2020-10-14T18:07:38.850+0800    INFO    controller-runtime.controller   Starting EventSource    {"controller": "guestbook", "source":
2020-10-14T18:07:38.953+0800    INFO    controller-runtime.controller   Starting Controller     {"controller": "guestbook"}
2020-10-14T18:07:38.953+0800    INFO    controller-runtime.controller   Starting workers        {"controller": "guestbook", "worker co
```

现在我们编辑生成的示例自定义资源 CR：

```
# config/samples/webapp_v1_guestbook.yaml
apiVersion: webapp.ydzs.io/v1
kind: Guestbook
metadata:
  name: guestbook-sample
spec:
  size: 2
  configMapName: test
  alias: Phone
```

然后在另外一个终端中安装上面的这个自定义资源：

```
$ kubectl apply -f config/samples/
guestbook.webapp.ydzs.io/guestbook-sample created
```

这个时候我们可以在控制器运行终端中看到对应的事件信息：

```
2020-10-14T18:09:54.461+0800    DEBUG   controller-runtime.controller   Successfully Reconciled {"controller": "guestbook", "request":
```

如果要将控制器部署到集群中去，首先需要构建并推送镜像到镜像仓库：

```
$ make docker-build docker-push IMG=<some-registry>/<project-name>:tag
```

根据 `IMG` 指定的镜像将控制器部署到集群中:

```
$ make deploy IMG=<some-registry>/<project-name>:tag
```

要从你的集群中删除 CRD 也很简单：

```
$ make uninstall
```