Navigating Kubernetes Custom Resources using code generation

Marta Paciorkowska

Go Oslo User Group, 4th Dec 2019

01

k8s + go = benefits

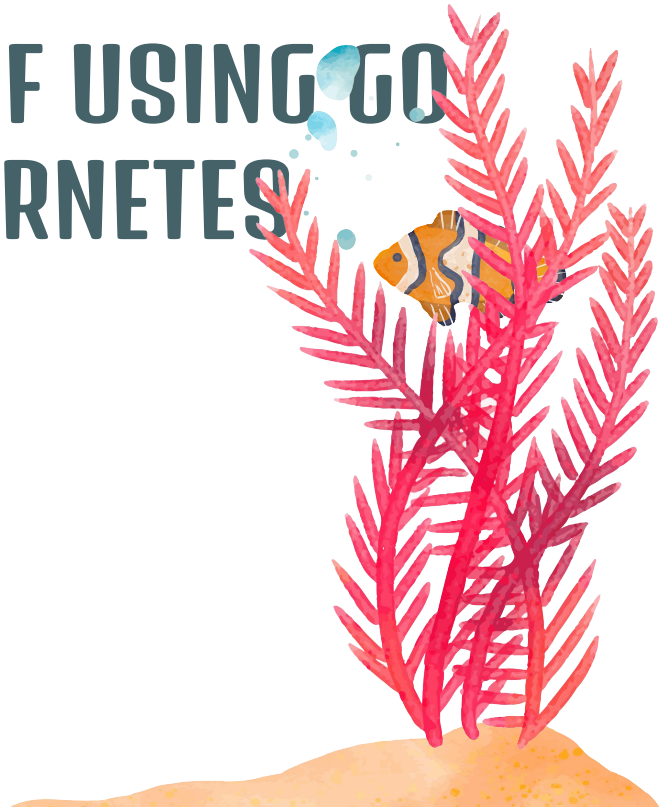
02

extending built-in resources

03

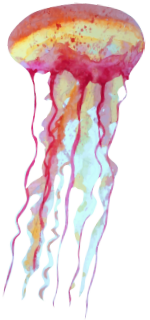
walkthrough

01 BENEFITS OF USING GO WITH KUBERNETES



Access to native K8s API	Pretty good performance	Statically typed
Have access to error types, etc.	Compiled to machine code. Comparably short compilation time.	Control over data structure gives safety.

02 EXTENDING BUILT-IN RESOURCES



K8S HAS SOME BUILT-IN TYPES...

Deployment

A set of identical Pods that get automatically replaced on failure.

Horizontal Pod Autoscaler

Automatically scales the number of Pods based on observed resource utilization.

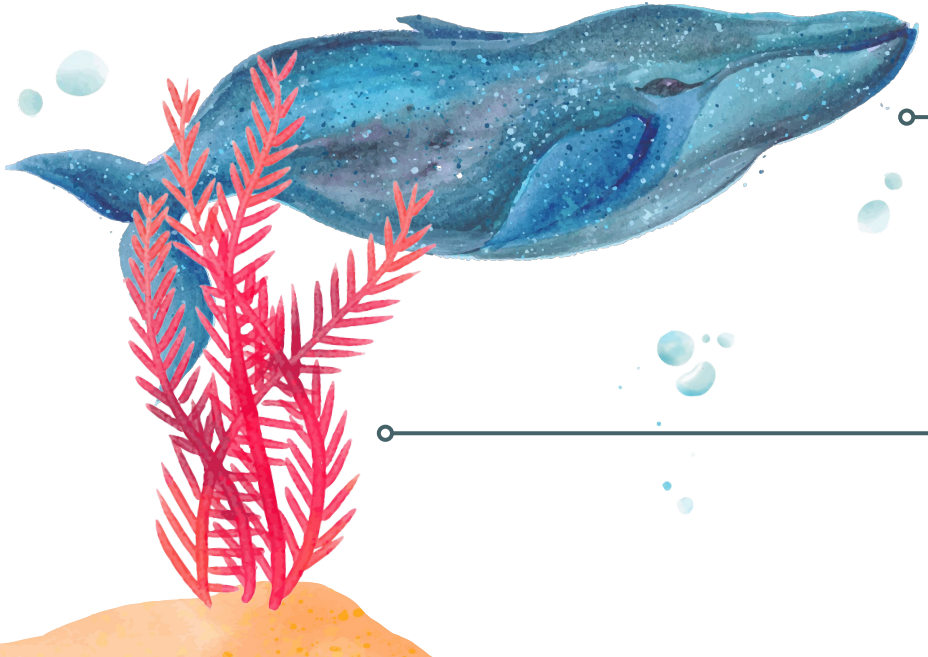
Service

A load balancer for a group of pods.

Ingress

Defines HTTPS endpoints for your Services.

...AND YOU CAN DEFINE YOUR OWN



A **Custom Resource** is an extension of the API.

A **custom controller** treats such a resource as desired state, and maintains it.

Examples: Postgres operator, fiaas-deploy-daemon.

03 WALKTHROUGH

We'll create a Custom Resource and a basic custom controller, using fiaas-deploy-daemon as inspiration.

PROCESS

STEP 1

Set up your project

STEP 2

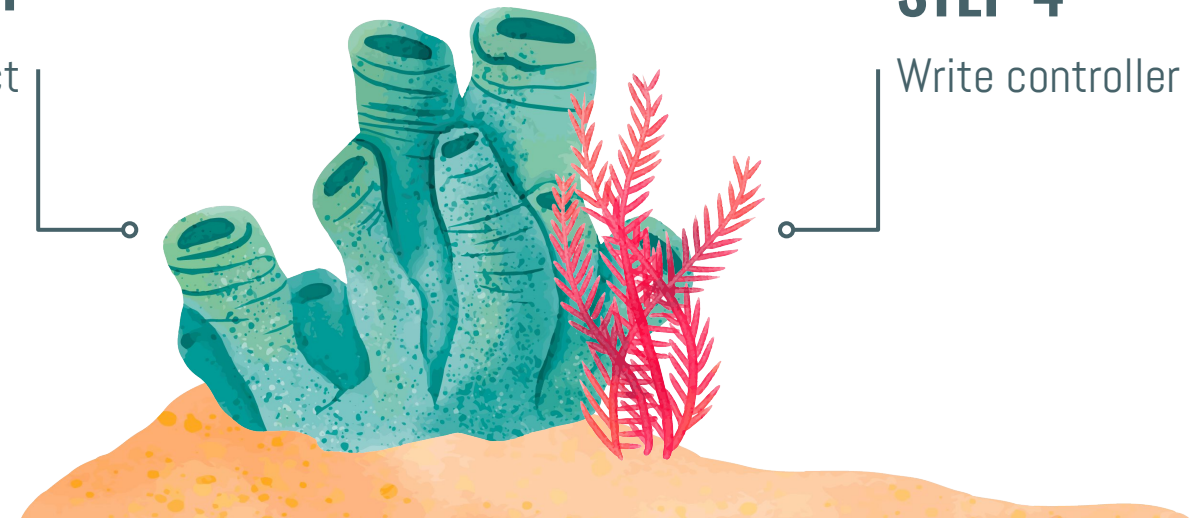
Define types & comments

STEP 3

Run generation script

STEP 4

Write controller



PROCESS

STEP 1

Set up your project

STEP 2

Define types & comments

STEP 3

Run generation script

STEP 4

Write controller

PRACTICAL PROJECT STRUCTURE

```
.
├── cmd
│   └── kubernetes-crd-demo
│       └── main.go
├── fixtures
│   ├── applicationA.yaml
│   ├── applicationB.yaml
│   └── crd.yaml
├── go.mod
├── go.sum
├── hack
│   ├── boilerplate.go.txt
│   ├── update-codegen.sh
│   └── verify-codegen.sh
├── LICENSE
├── Makefile
├── minikube
├── pkg
│   └── apis
│       └── gooslo.io
│           └── v1
│               ├── crd.go
│               ├── doc.go
│               ├── register.go
│               └── types.go
└── README.md
```

Output of `tree`

PROCESS

STEP 2

Define types & comments

STEP 3

Run generation script

STEP 1

Set up your project

STEP 4

Write controller

TYPE/DATA STRUCTURE

Our example: a simple Application Custom Resource that can help us abstract four built-in types mentioned earlier:

1. **Application**

   The top level type. It has an...

2. **ApplicationSpec**

   This struct has only two fields: **Image** and **Name**.

A NOTE ON MAGIC COMMENTS

Let you control **what** will be generated.

Note that the full list is hard to find. One person aggregated a list of code gen tags.

What we'll use today:

`// +genclient` - generate all client verb functions

`// +genclient:nonNamespaced` - generate without namespace

`// +groupName=gooslo.io` – used in the fake client as the full group name, defines the fully qualified API group name.

`// +k8s:deepcopy-gen:interfaces` - used in cases where you define API types that have fields of some interface type.

PROCESS

STEP 1

Set up your project

STEP 2

Define types & comments

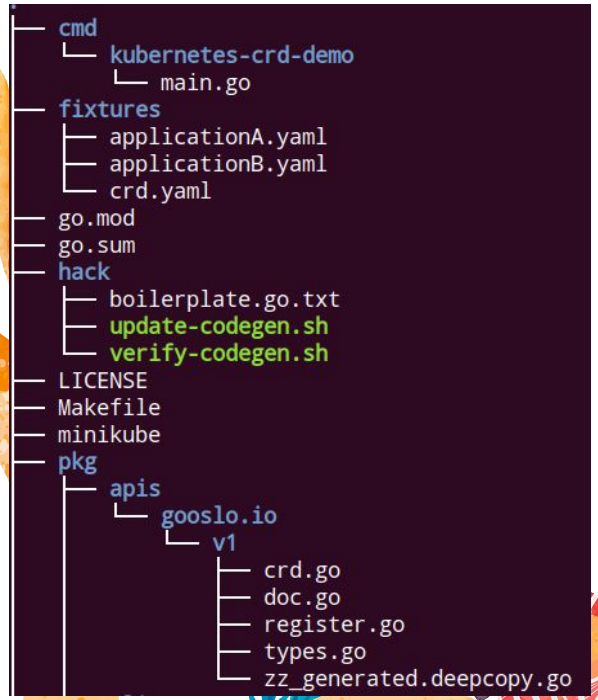
STEP 3

Run generation script

STEP 4

Write controller

AFTER GENERATION SCRIPT



DEEPCOPY FUNCTIONS AND INTERFACES

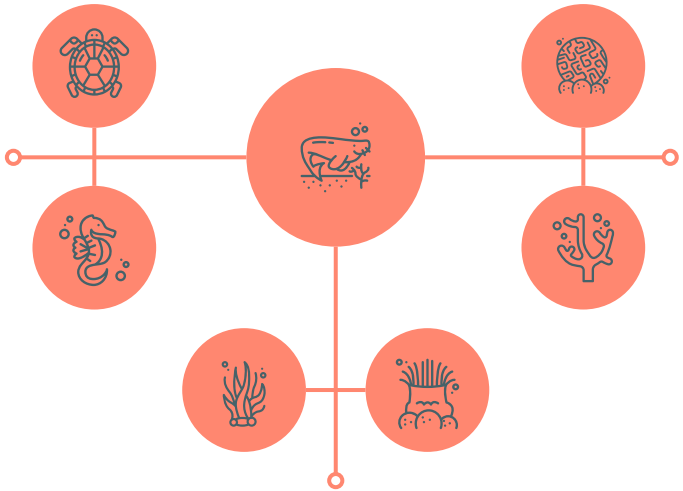
Custom Resources in Go have to implement the `runtime.Object` interface.

This in done using deep copy functions, generated by the `deepcopy-gen` package.

AVAILABLE COMPONENTS

LISTERS

list and get your custom resource



INFORMERS

one way of caching for processing

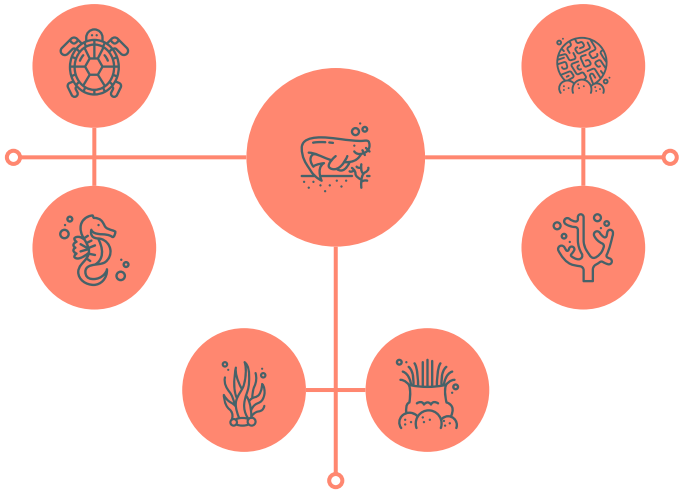
CLIENTSET

your master key

AVAILABLE COMPONENTS

LISTERS

list and get your custom resource



INFORMERS

one way of caching for processing

CLIENTSET

your master key

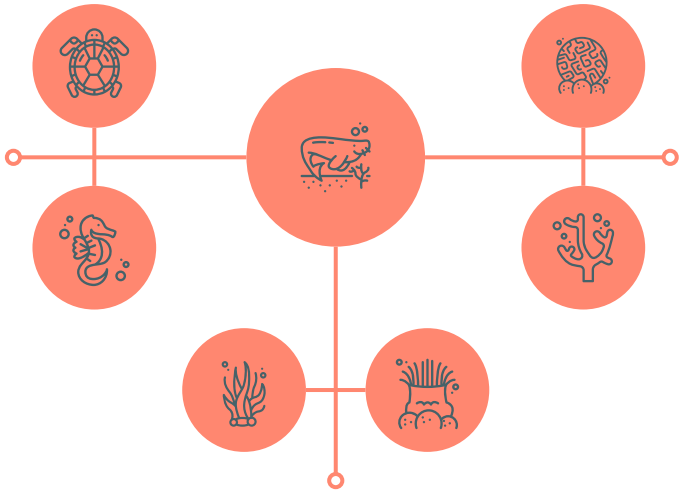
HOW DO CLIENTSETS WORK?

Resources are members of API groups (`core`, `extensions`).

Groups have versions (`core/v1`, `extensions/v1beta`)

Clients for each versioned group are collected in a clientset.

AVAILABLE COMPONENTS



LISTERS

list and get your custom resource

INFORMERS

one way of caching for processing

CLIENTSET

your master key

WHAT'S AN INFORMER?

Informs about resource
updates using an
event-based interface.

PROCESS

STEP 1

Set up your project

STEP 2

Define types & comments

STEP 3

Run generation script

STEP 4

Write controller

USING OUR NEW CRD CLIENTSET

The controller uses an infinite loop that watches for changes (events) to our Application custom resource.

It then prints out the type of event, name of the application, and what Docker image it should run.

HOW DO WE TEST THE CODE?

Our generated code has a fake client that can be used to mock the real objects. From Naiserator:

```
appClient := nais_fake.NewSimpleClientset()

app, err := appClient.
    NaiseratorV1alpha1().
    Applications(namespace).
    Create(app)
…
persistedApp, err := appClient.
    NaiseratorV1alpha1().
    Applications(namespace).
    Get(name, metav1.GetOptions{})
```

THANK YOU!

Does anyone have any **questions**?

This code lives at:

github.com/xamebax/kubernetes-crd-demo

Contact me:

mpaciorkowska @ Go slack
a_meba @ Twitter

CREDITS

- Presentation template by Slidesgo
- Icons by Flaticon
- Infographics by Freepik
- Author introduction slide photo created by Freepik
- Text & Image slide photo created by Freepik.com