Controller: Extending your K8s cluster

Terin Stock
Ross Guarino

Introductions

Terin Stock

@terinjokes

Ross Guarino

@0xRLG

Kubernetes at Cloudflare

Pre-Kubernetes at Cloudflare

Code Review

Salt

JIRA

Emails

Laptop

Mesos

Pre-Kubernetes at Cloudflare

Low Cohesion

&

High Coupling

Keeping it Simple

- Automate processes
    - Make correct the easiest
- Abstract implementation
    - Move the decision making elsewhere
- Remove duplicate state
    - Say it only once

Kubernetes Out of the box

We run all of our own physical infrastructure.

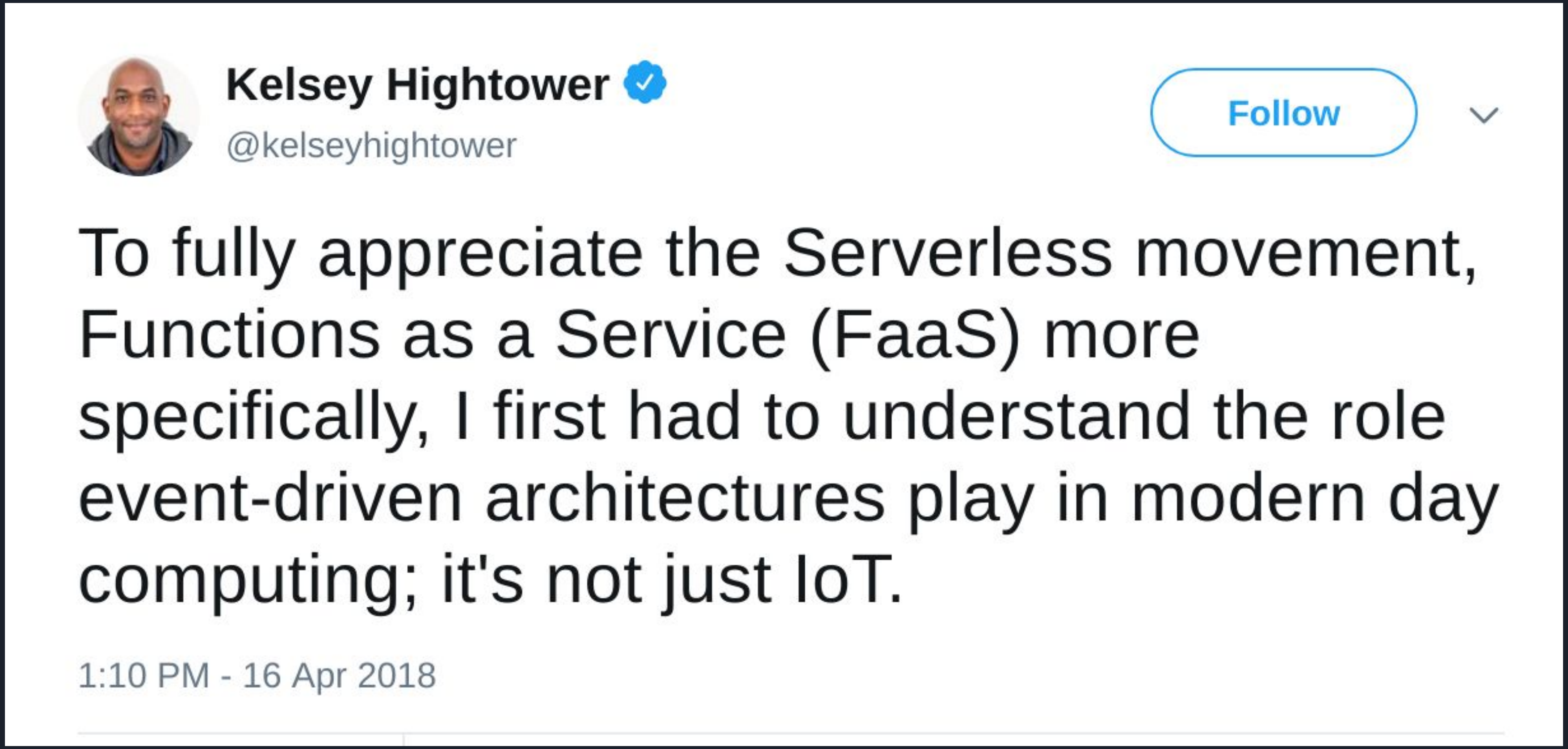
So, Kubernetes the Salt way is the only option

Mind the Gap

Without a Cloud Provider...

- No cloud load balancer
  - Leifur: a load balancer for bare metal
- Hardware scales much slower
  - Pyli: automate user and namespace creation and RBAC provisioning
- Existing telemetry
  - Rule Loader: configure Prometheus from ConfigMaps

Can’t believe it’s not Serverless!



Kelsey Hightower
@kelseyhightower

To fully appreciate the Serverless movement, Functions as a Service (FaaS) more specifically, I first had to understand the role event-driven architectures play in modern day computing; it's not just IoT.

1:10 PM - 16 Apr 2018

Can’t believe it’s not Serverless!

Controllers are:

- Simple
- Reliable
- Event driven
- Easy to write

Example Problem

We want a namespace for every developer on the Kubernetes cluster.

Possible Solutions:

- Offload it to the IT department
- Onboarding tasks the new hire does their first week
- Write a standalone service

Or, Writing a Controller

We can write a controller which maintains the relationship between a User and a Namespace

4 Steps to Writing a Controller

1. Define your Custom Resource Definition
2. Generate Client Code
3. Listen for events
4. Handle events in queue

Define a Custom Resource

Creating a Custom Resource for Users

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: users.example.com
spec:
  group: example.com
  version: v1
  scope: Cluster
  names:
    plural: users
    singular: user
    kind: User
    shortNames:
    - usr
```

Validating Objects

- Resources can be checked against OpenAPI v3 schema on admission

K8s Code Gen

Generating Client Code

github.com/kubernetes/code-generator

- Client Code
- Informers
- Listers
- DeepCopy

pkg/apis/example.com/v1/types.go

```
// +genclient
// +genclient:noStatus
// +k8s:deepcopy-gen=true
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type User struct {
	metav1.TypeMeta   `json:",inline"`
	metav1.ObjectMeta `json:"metadata,omitempty"`
	Spec UserSpec`json:"spec"`
}

// +k8s:deepcopy-gen=true
type UserSpec struct {
	DisplayName string `json:"display_name"`
}
```

Installing the Generator

Add the code generator to your Gopkg.toml file:

```
required = ["k8s.io/code-generator/cmd/client-gen"]

$ dep ensure
```

Running the generator

```
$ ./vendor/k8s.io/code-generator/generate-groups.sh \
  all \
  example.com/pkg/clientexample.com/pkg/apis \
  example.com:v1
```

```
pkg/
└── apis
    └── example.com
        └── v1
            ├── docs.go
            ├── register.go
            └── types.go
```

→

```
pkg/
├── apis
│   └── example.com
│       └── v1
│           ├── docs.go
│           ├── register.go
│           ├── types.go
│           └── zz_generated.deepcopy.go
└── client
    ├── clientset
    │   └── [...]
    ├── informers
    │   └── [...]
    └── listers
    │   └── [...]
```

Listening for Events

Informers

- React to the changes in resources
- Reduce the burden on API server
- Populate read-only cache (Lister)
- Prevents polling

Listers

- Read-only cache populated by Informers
- Reduce burden on API server

Work Queues

Simple, Intelligent Workqueue:

- Stingy
- Fair
- Multiple Consumers and Producers

What goes on the queue?

1. Queues use equivalent to determine duplicate keys
2. The simpler the objects the better
3. Usually .metadata.name works well

```go
queue := workqueue.NewRateLimitingQueue()
informers := informers.NewSharedInformerFactory(
    clientSet,
    time.Second * 30
)

func enqueueUser(queue workqueue.Type, obj interface) {
    key, _ := cache.DeletionHandlingMetaNamespaceKeyFunc(obj)
    queue.Add(key)
}
```

```
informers.Example().AddEventHandler(
    &cache.ResourceEventHandlerFuncs{
        AddFunc: func(obj interface{}) error {
            return enqueueUser(queue, obj)
        },
        UpdateFunc: func(_, obj interface{}) error {
            return enqueueUser(queue, obj)
        },
        DeleteFunc: func(obj interface{}) error {
            return enqueueUser(queue, obj)
        },
    })
```

Watching Children

- Update the child's metadata.ownerReferences to reflect the relationship
- In our case we want to be notified if namespace we care about changes.

Handling Events

Worker go routine

- Pops items off of the queue and calls a work function until instructed to stop
- Cannot block forever on one item

Work Functions:

- Handle Deletion
- Idempotent

```go
func processWorkItem(
    queue workqueue.Delaying,
    workFn func(context.Context, string) error
) {
    // get the item or signal to quit
    key, quit := q.Get()
    if quit {
        return false
    }
    defer q.Done(key) // Tell the queue we’re done processing this item
    ctx, cancel := context.WithTimeout(context.Background(), 30*time.Second)
    defer cancel()

    err := workFn(ctx, key.(string))
    if err == nil {
        q.Forget(key) // Mark the work as successful
        return true
    }
    q.AddRateLimited(key) // Retry at a later time
)
```

Tips & Tricks

Don’t handle Transient Errors

- Don’t panic, just return
- Return errors but don’t retry in your worker functions
- Let the queue retry them

Handling Deletion

Do you have external state?

Do you need to guarantee you witness a deletion?

Don’t Handle the OnDelete Differently

Avoid duplicating & complicating your code.

Consider this a best-effort optimization opportunity for later.

No? Use Kubernetes Garbage Collection

“[The garbage collector will delete] objects that once had an owner, but no longer have an owner.”

There’s no code to write! Since we’ve already set up ownerReferences for notifications.

Yes? Use Finalizers for deletion

- Don’t rely on noticing the deletion event
- Use a finalizer to handle deletions

How do finalizers work?

When you delete a resources with Finalizers Kubernetes will wait until all existing Finalizers are removed then finally delete the resources.

On resource deletion, Kubernetes waits for each Finalizer to complete before removing the resource.

```
func syncUser(
    key string,
    client exampleclient.ClientSet,
    userLister listers.UserLister,
    k8sClient kuberentes.ClientSet,
    nsLister lister.NamespaceLister
) error {
    // Get the User from the cache
    cached, _ := userLister.Get(key)

    if cached.DeletionTimestamp.IsZero() &&
       apiextensions.CRDHasFinalizer(cachedCRD, "example.com") {
       // HANDLE DELETE
       // Remove example.com from Finalizer list
    }
    // HANDLE UPDATE/CREATE
}
```

TAK, QUESTIONS?