Write your first Kubernetes controller in less than 3 hours

Introduction

Federico Paolinelli

- Telco Network Team @ Red Hat
- All things networking and kubernetes
- MetalLB maintainer

Francesco Romani

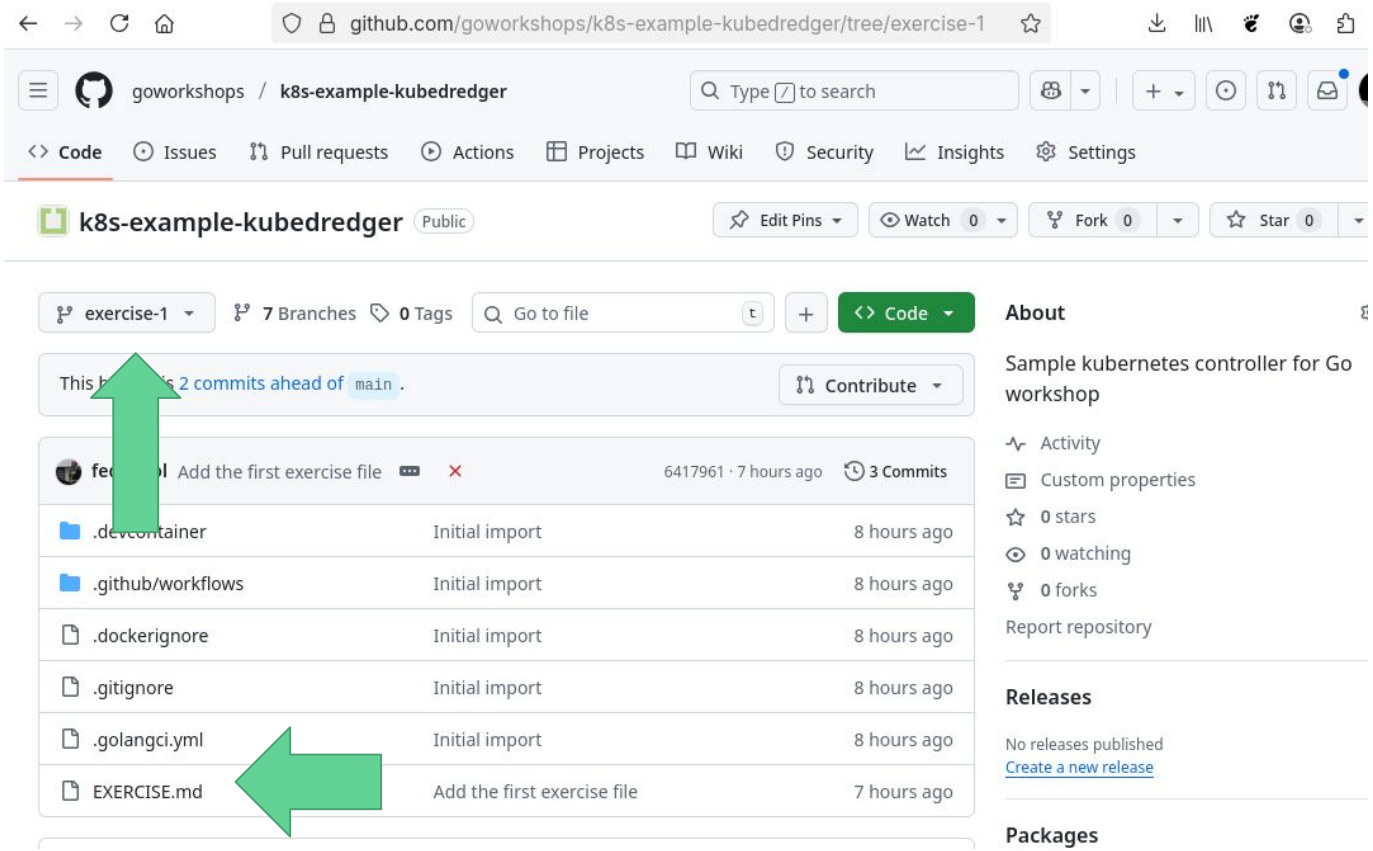
- Telco Compute Team @ Red Hat
- Kubernetes tuning/enhancement for low latency
- Kubernetes SIG-node maintainer

Workshop structure

- Topic introduction
- Practice
- Practice review

github.com/goworkshops/k8s-example-kubedredger

github.com/goworkshops/k8s-example-kubedredger

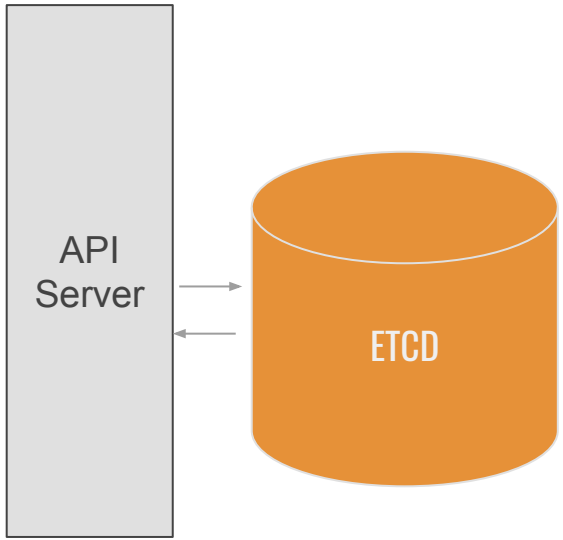


The Kubernetes model

Kubernetes deals with objects

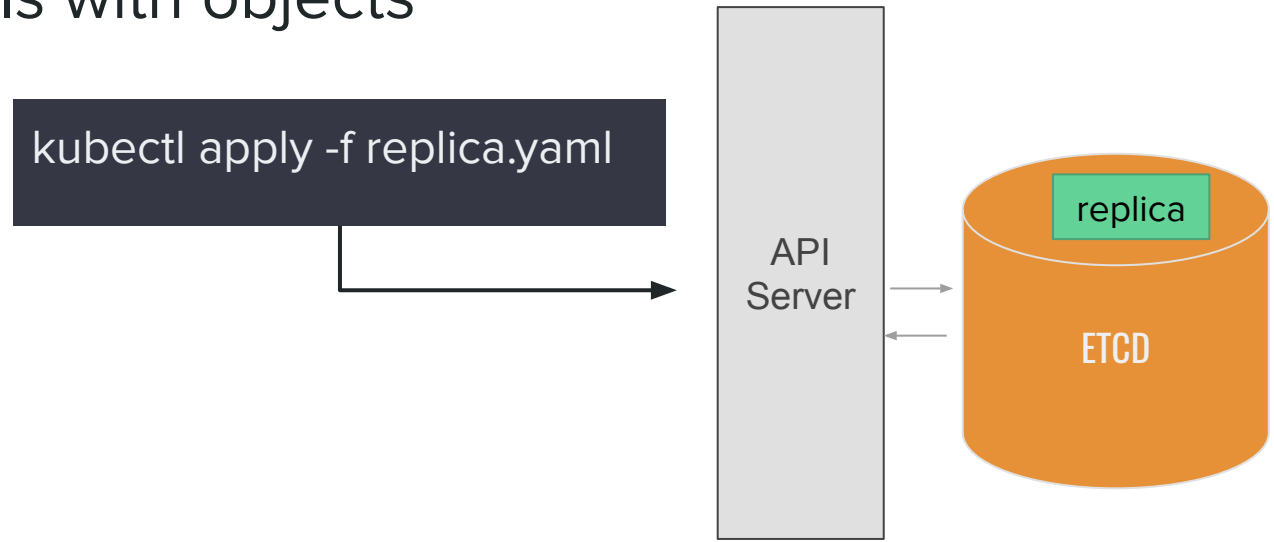
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

```
kubectl apply -f replica.yaml
```



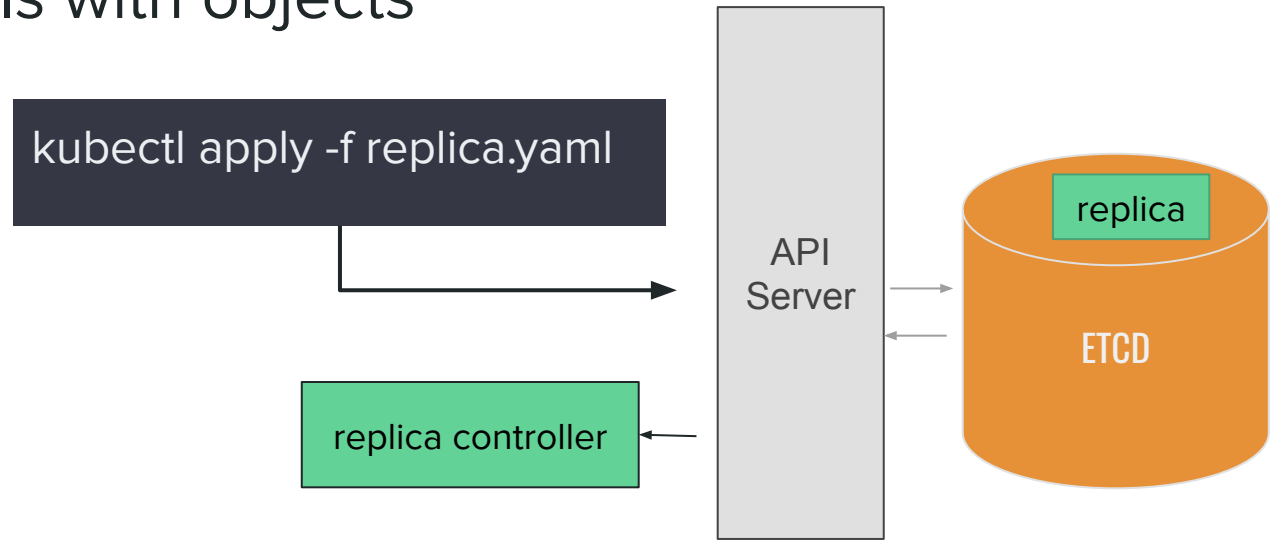
Kubernetes deals with objects

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```



Kubernetes deals with objects

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```



Declarative over imperative

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

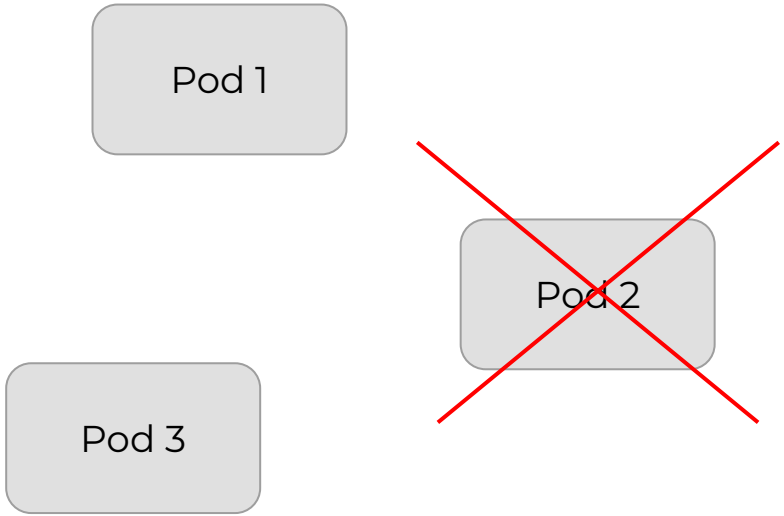
Pod 1

Pod 2

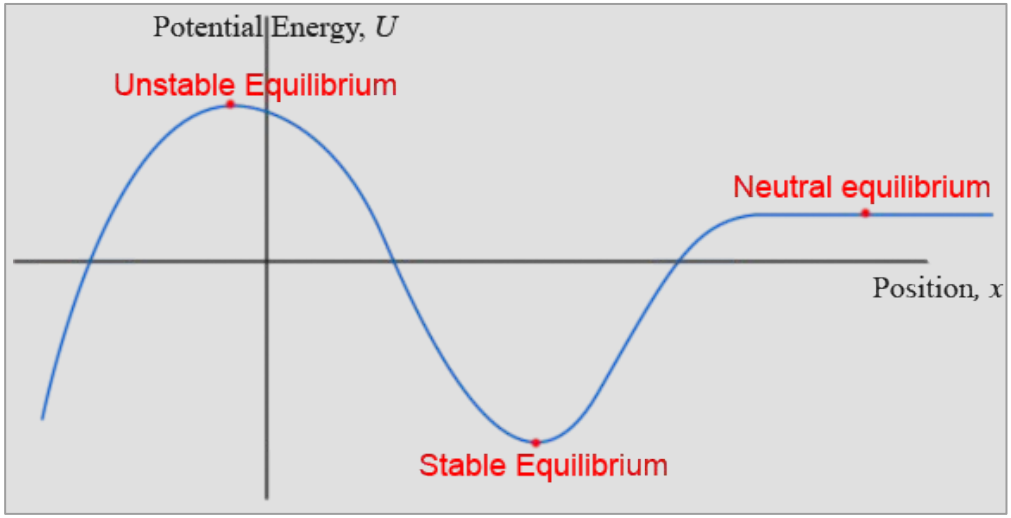
Pod 3

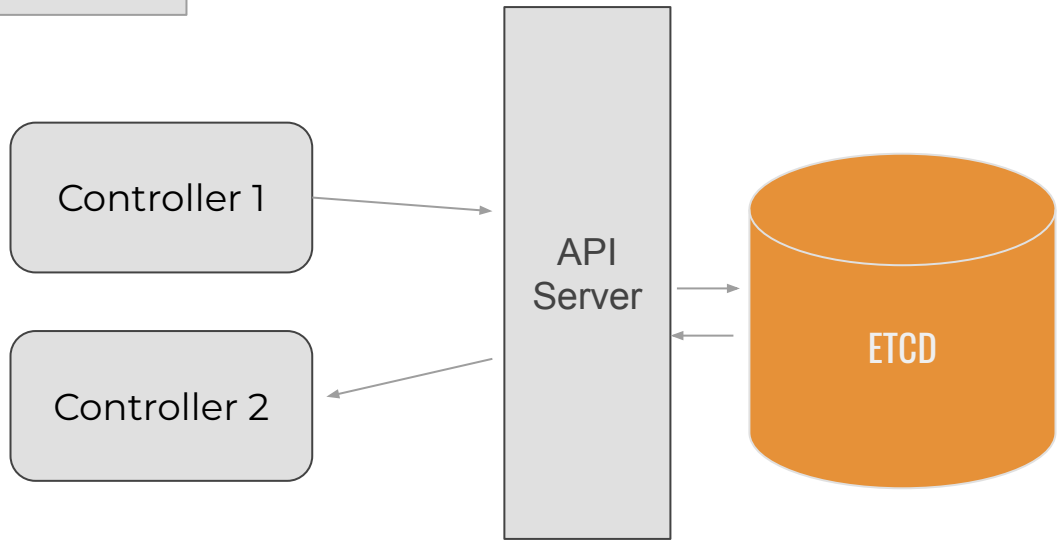
Declarative over imperative

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```



A Kubernetes cluster is a dynamic system


Potential Energy, U
Unstable Equilibrium
Neutral equilibrium
Position, x
Stable Equilibrium

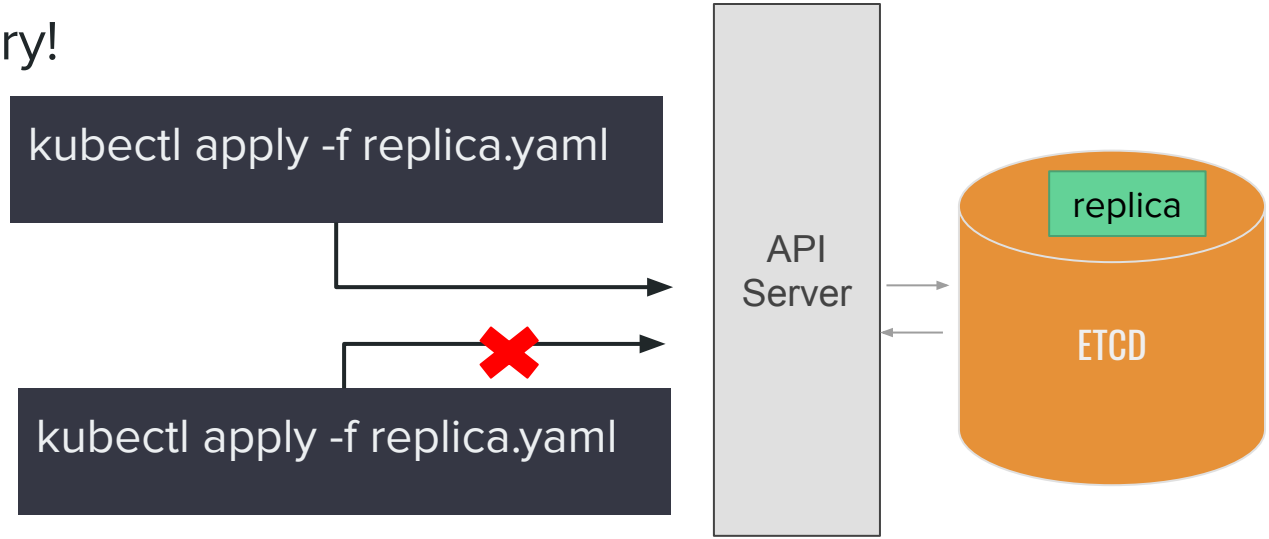

Controller 1
Controller 2
API Server
ETCD

Complications

https://flic.kr/p/HCWbEi

Knock knock! Race condition! Who's there?

- Kubernetes is read intensive
- Concurrency is handled optimistically
- Conflicts are resolved with failures
- The client must retry!



Missed Events!

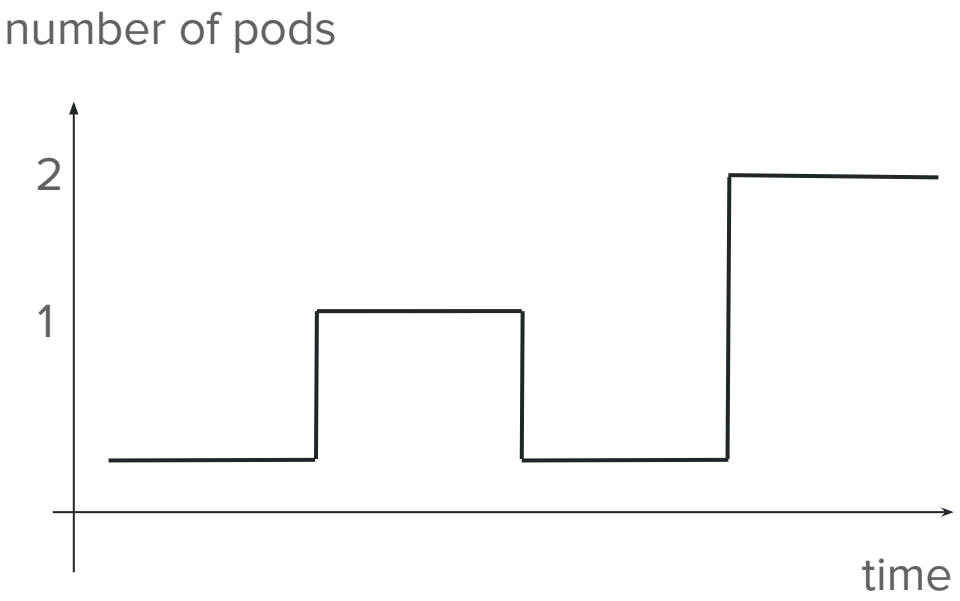
Your controller can be restarted at any time

The network is not reliable

The whole node can crash!

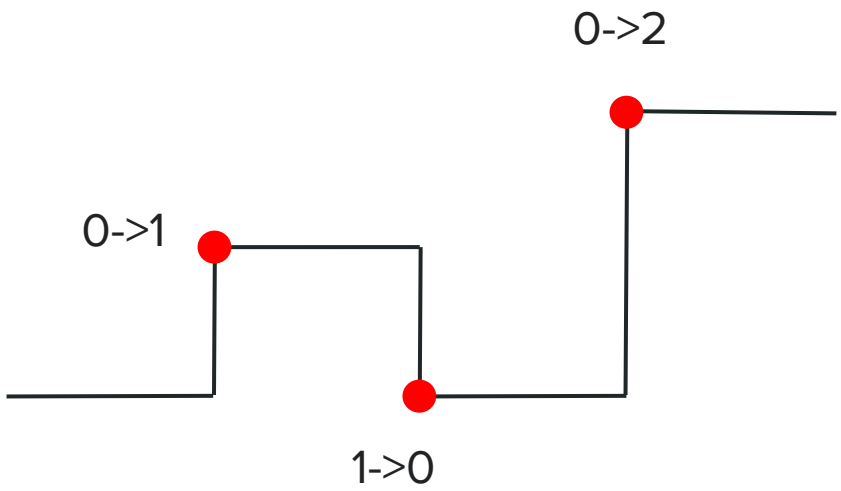
Edge driven - Level driven

Edge Driven vs Level Driven



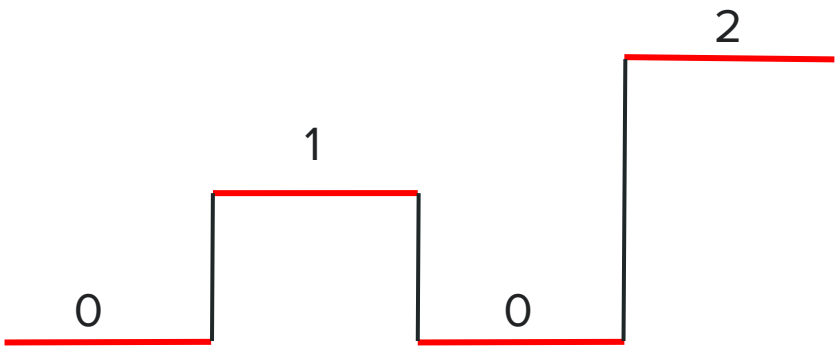
Let’s consider the variation of the number of pods in a deployment.

Edge Driven



When driven by edges, the behaviour depends on the variation of the data observed at edge location.

Leven Driven



With levels, we care about the level, in our case the state of the system at a given time.

Edge Driven vs Level Driven?

- Handling the variation might seem easier
- **Leven driven is more robust**
- The reconciliation logic is more complex

The theory

Objects

Kubernetes objects taxonomy

```
/apis/batch/v1/namespaces/FOO/jobs
```

Kubernetes objects taxonomy

- Group

```
/apis/batch/v1/namespaces/FOO/jobs
```

Kubernetes objects taxonomy

- Group
- Version

```
/apis/batch/v1/namespaces/FOO/jobs
```

Kubernetes objects taxonomy

- Group
- Version
- Resource

`/apis/batch/v1/namespaces/FOO/jobs`

All the objects share the same structure

Common traits

```
type Pod struct {
    metav1.TypeMeta
    metav1.ObjectMeta
    Spec   PodSpec
    Status PodStatus
}
```

TypeMeta

```
type TypeMeta struct {
    Kind       string
    APIVersion string
}
```

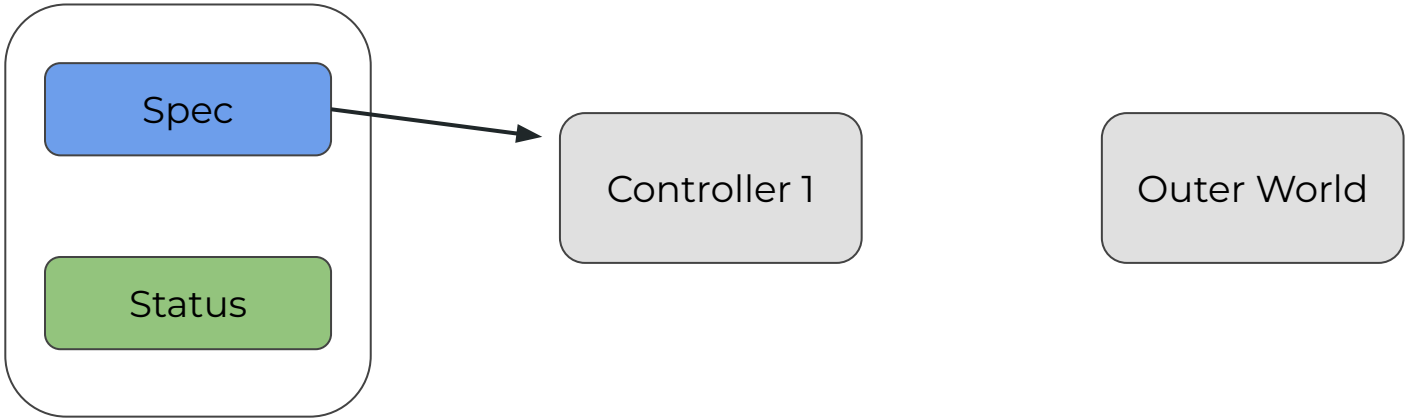
ObjectMeta

```
type ObjectMeta struct {
    Name                string
    GenerateName        string
    Namespace           string
    SelfLink            string
    UID                 types.UID
    ResourceVersion     string
    Generation          int64
    .
    .
```

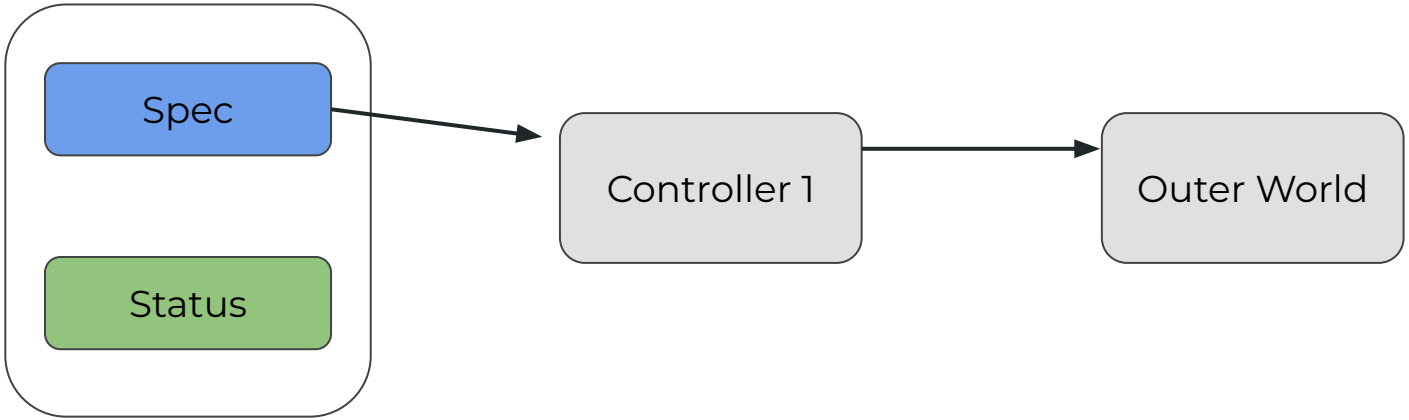
Spec and Status

```
type Pod struct {
    metav1.TypeMeta
    metav1.ObjectMeta
    Spec   PodSpec
    Status PodStatus
}
```

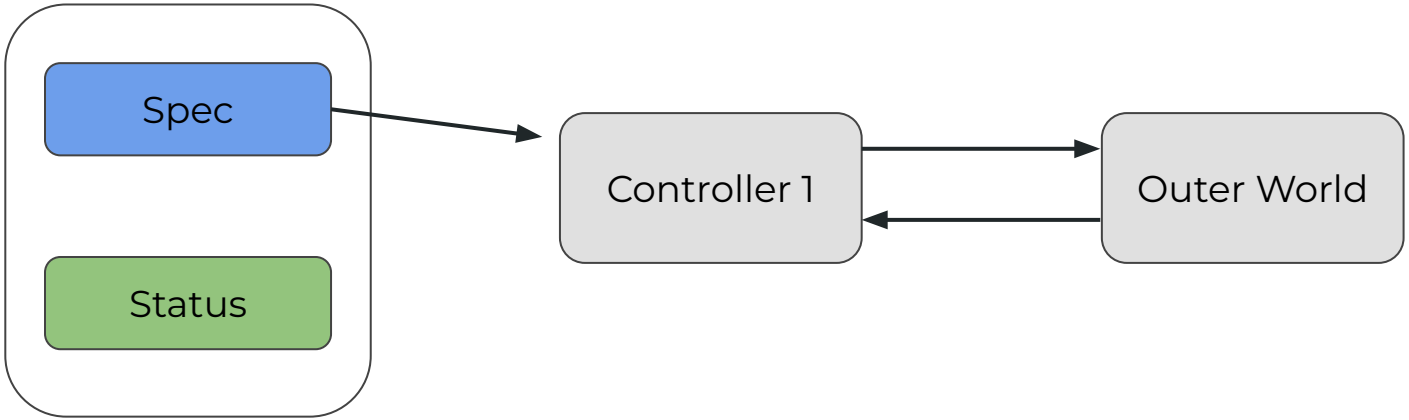
Spec is the desired state, status the actual state



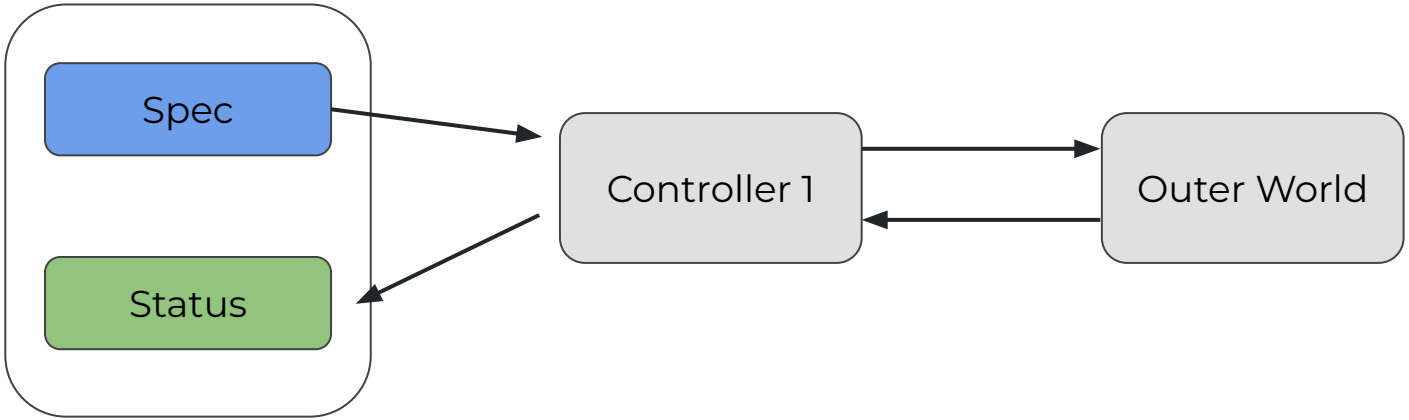
Spec is the desired state, status the actual state



Spec is the desired state, status the actual state



Spec is the desired state, status the actual state

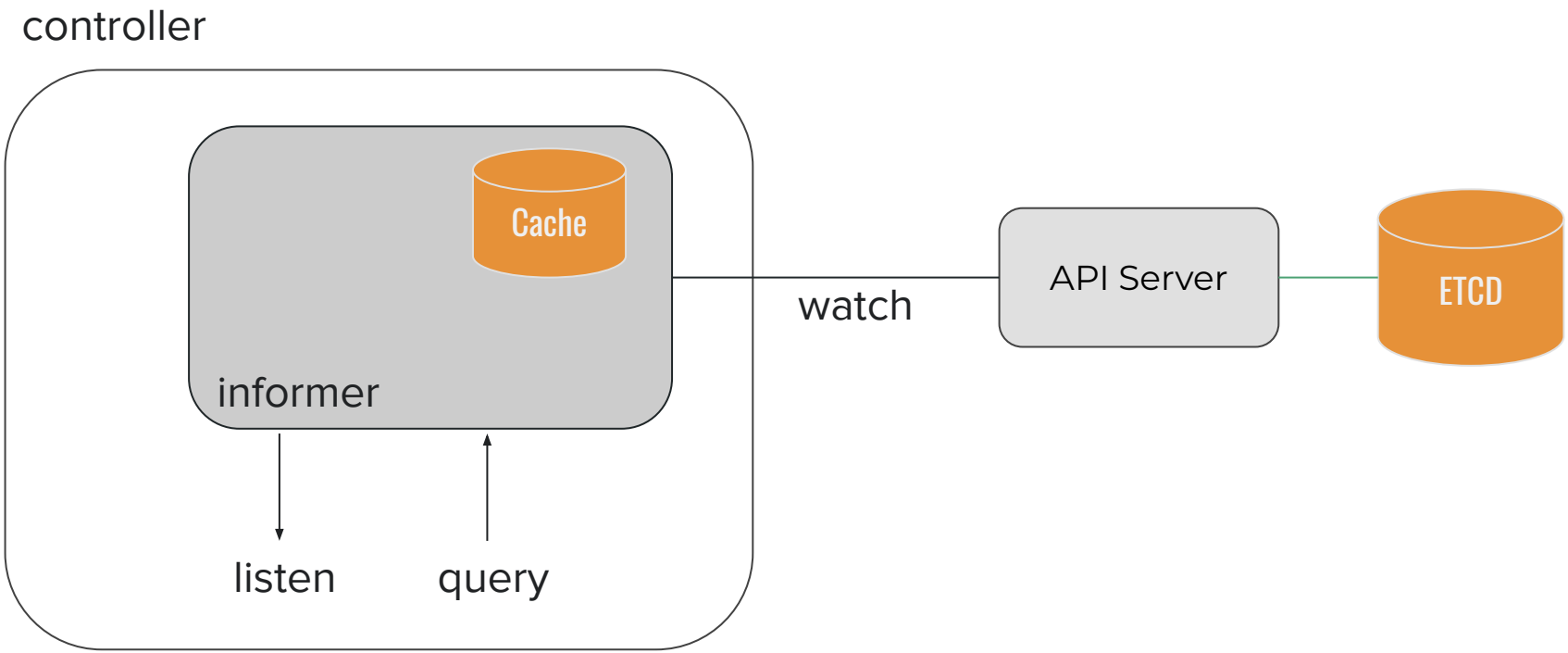


The building blocks

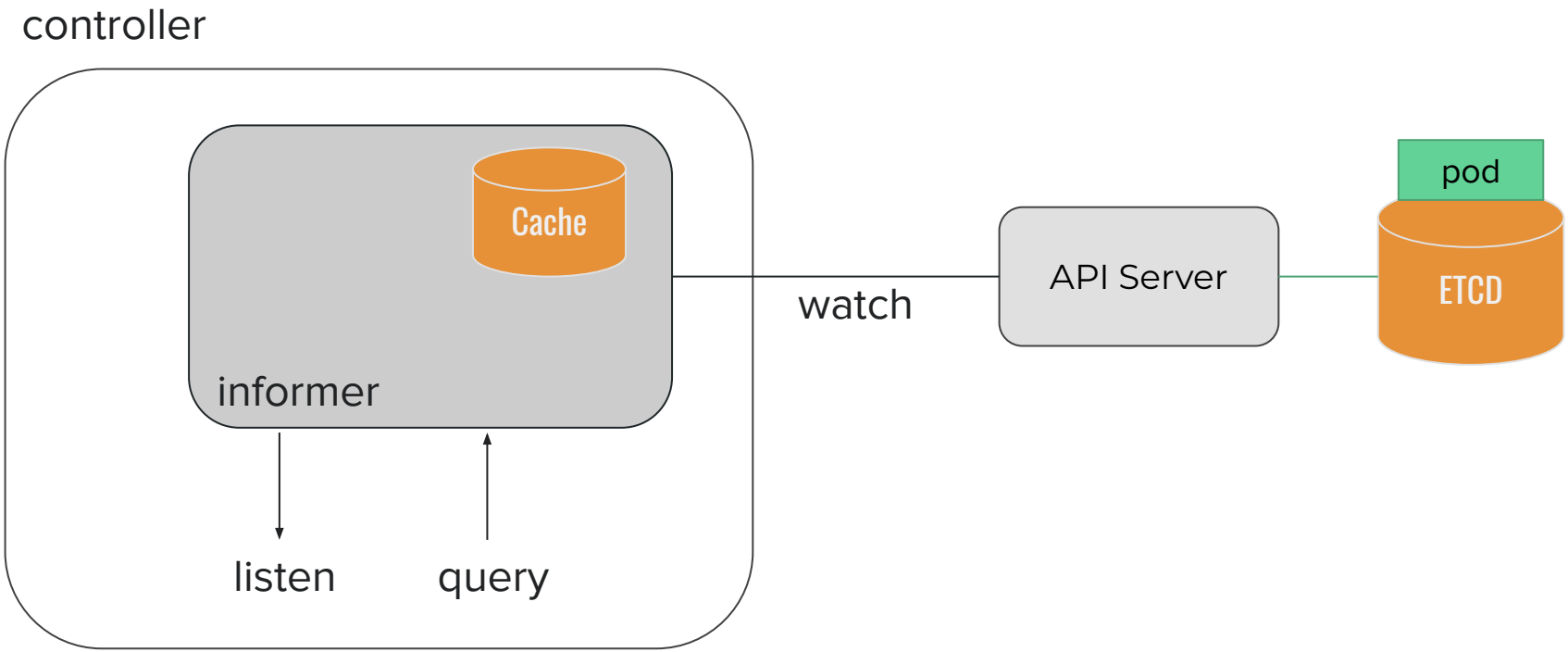
Informers

- Contain a local cache
- Can watch for resources
- Handles reconnections
- Can notify a client

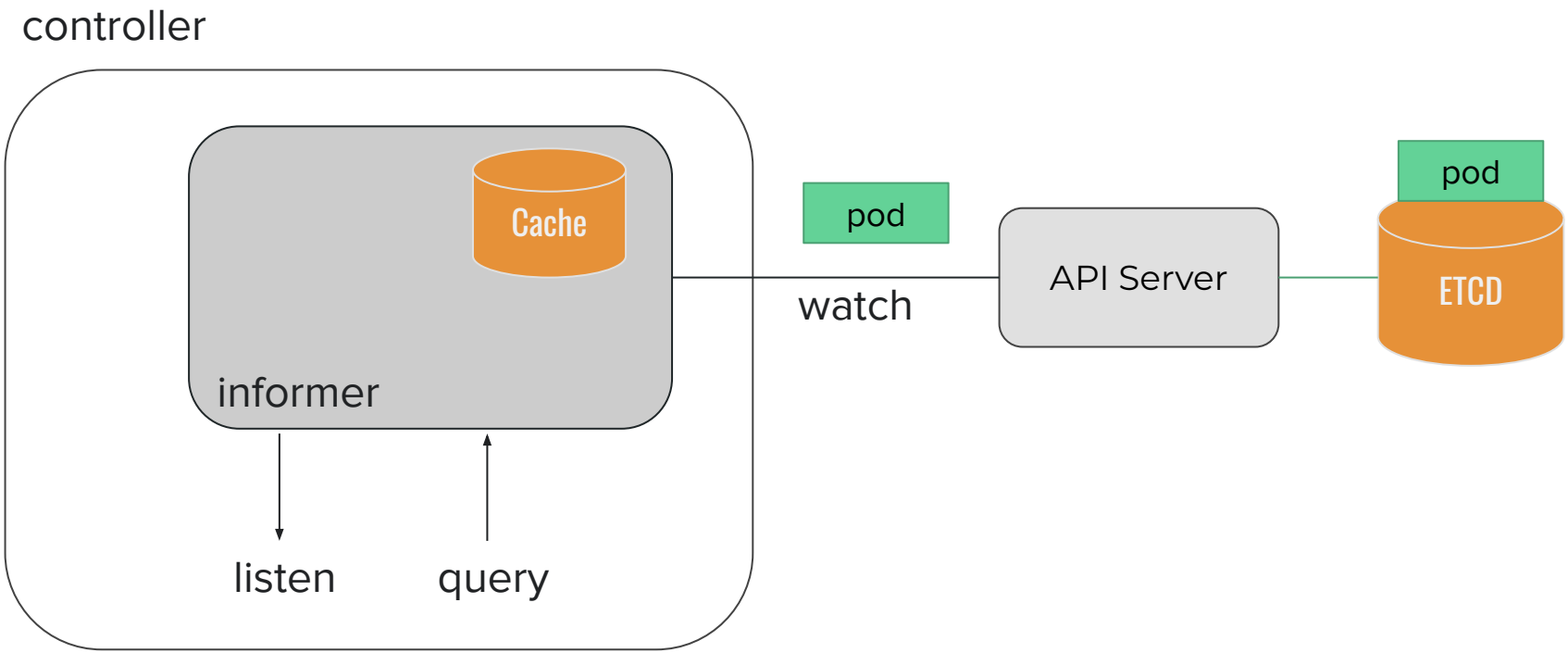
Informers


controller
Cache
informer
listen
query
watch
API Server
ETCD

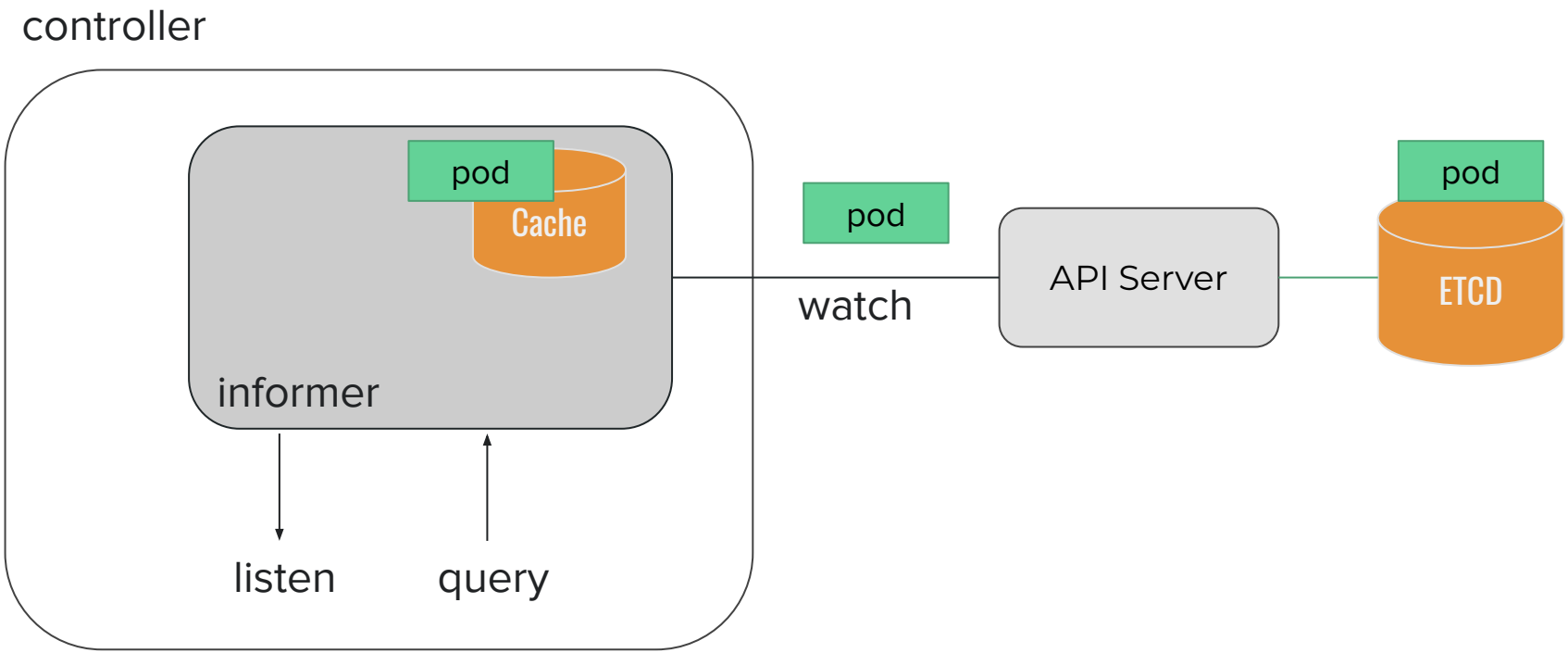
Informers


controller
Cache
informer
listen
query
watch
API Server
pod
ETCD

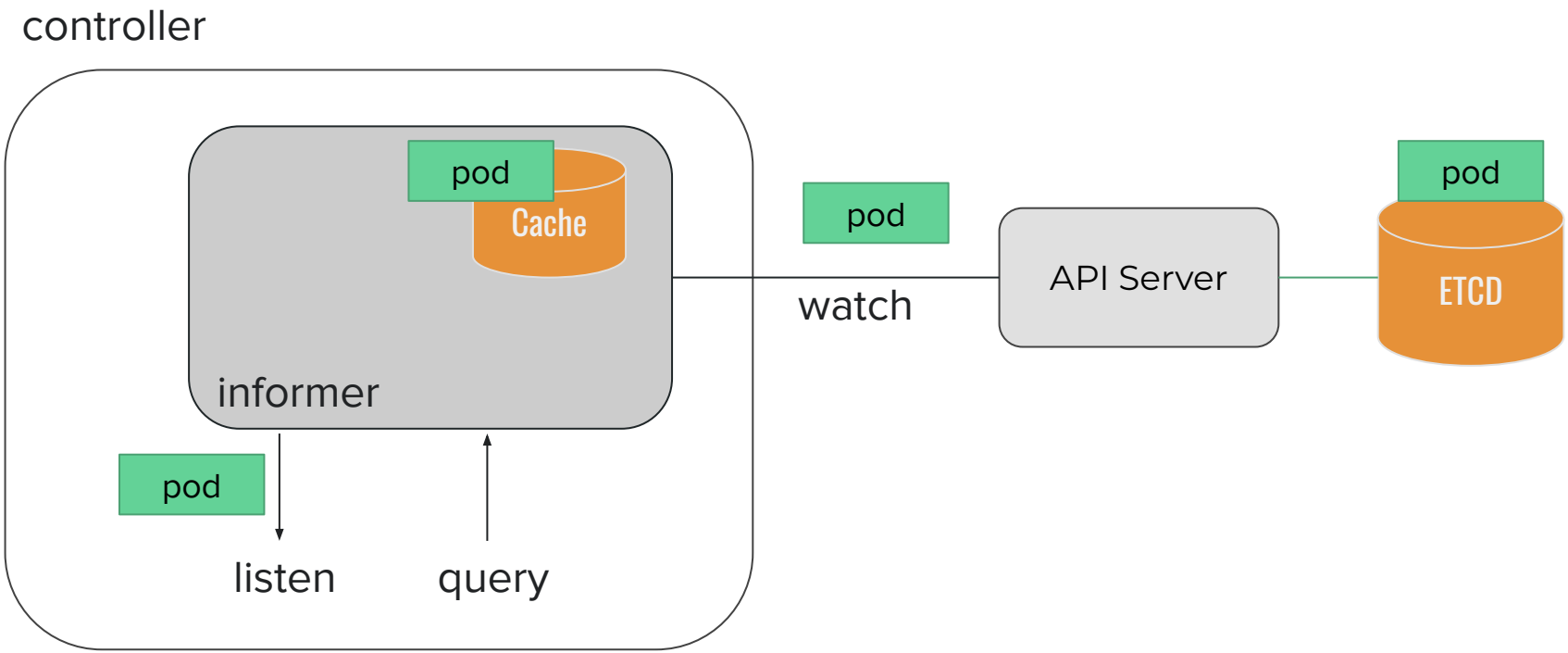
Informers


controller
Cache
informer
listen
query
pod
watch
API Server
pod
ETCD

Informers


controller
pod
Cache
informer
listen
query
pod
watch
API Server
pod
ETCD

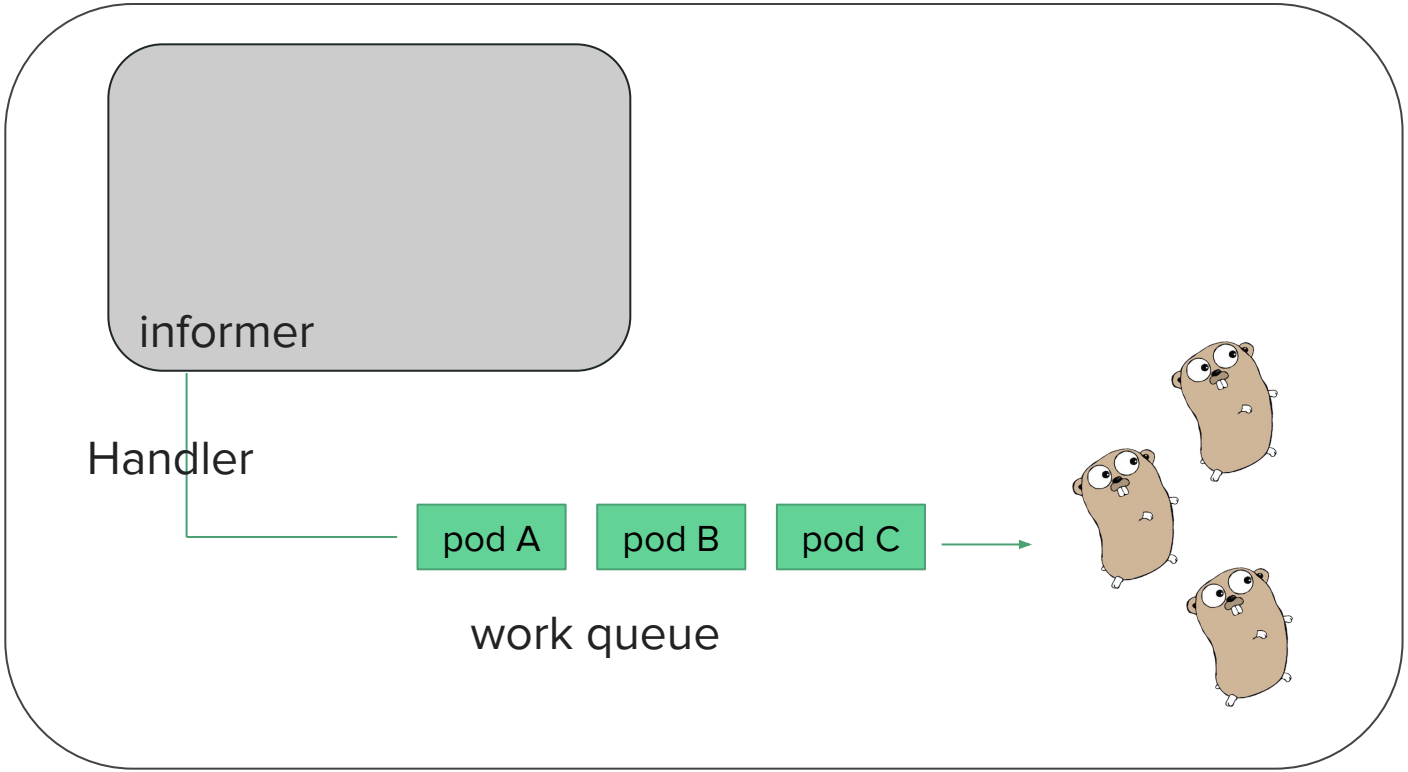
Informers


controller
pod
Cache
informer
pod
listen
query
pod
watch
API Server
pod
ETCD

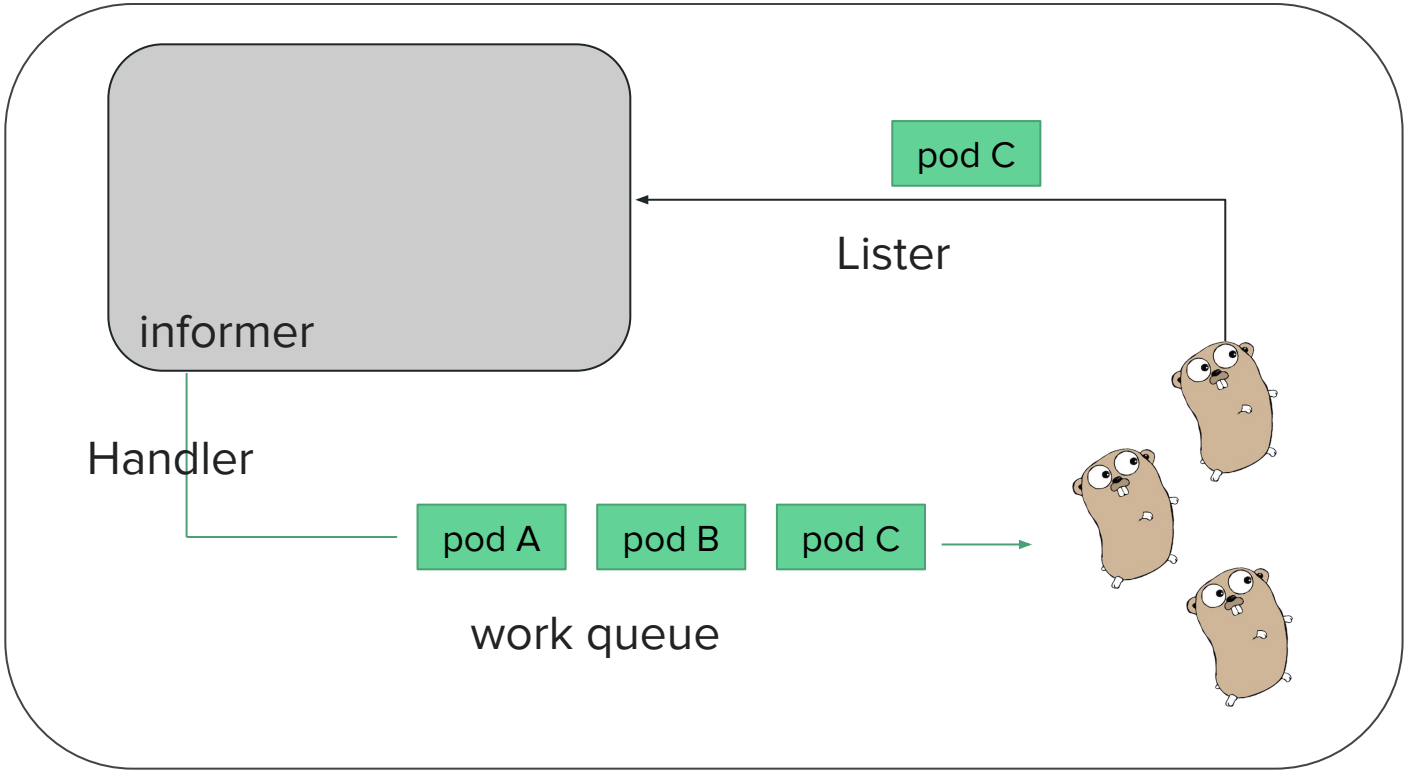
Work queues

- Queue implementation
- Contains only the key of the object
- A given key occurs only once
- In case of failure the key can be pushed back to the queue

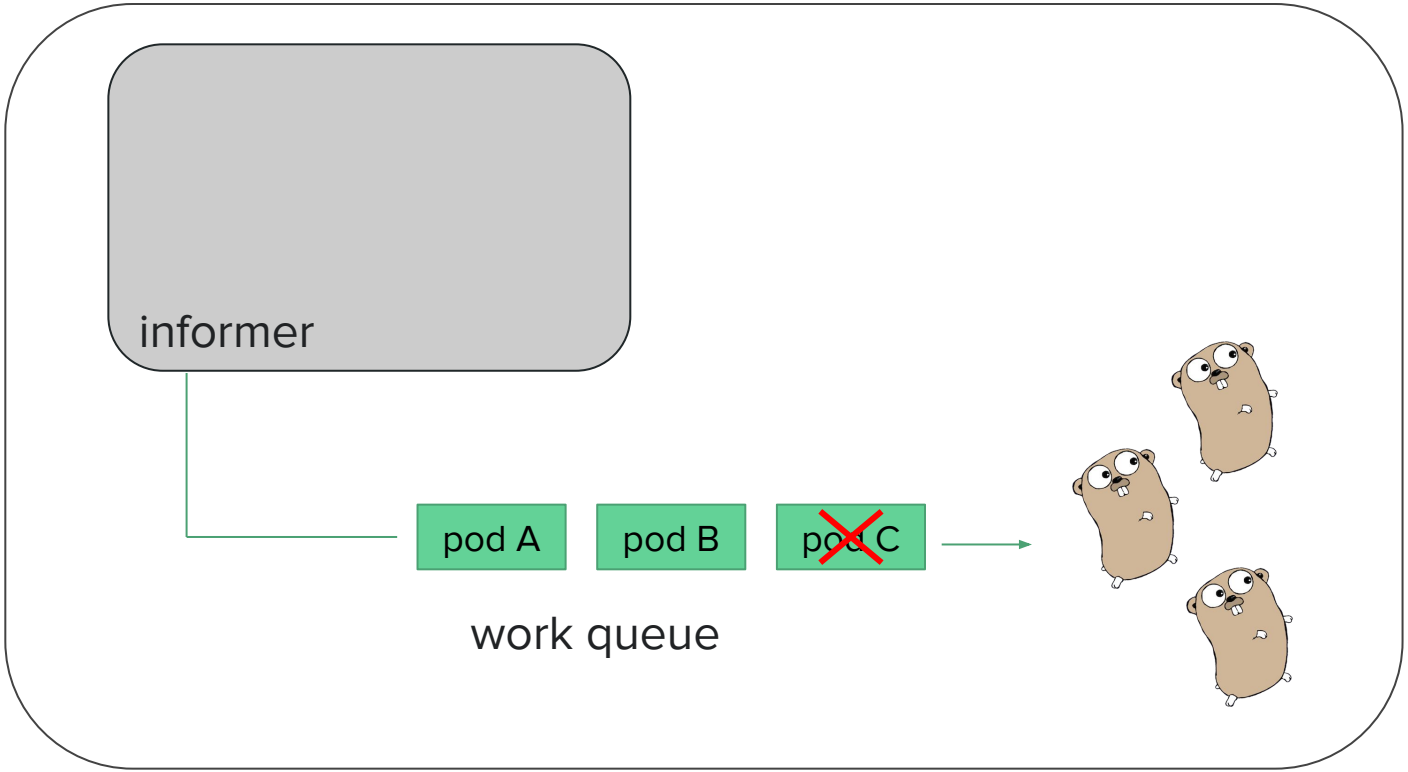
Work queues



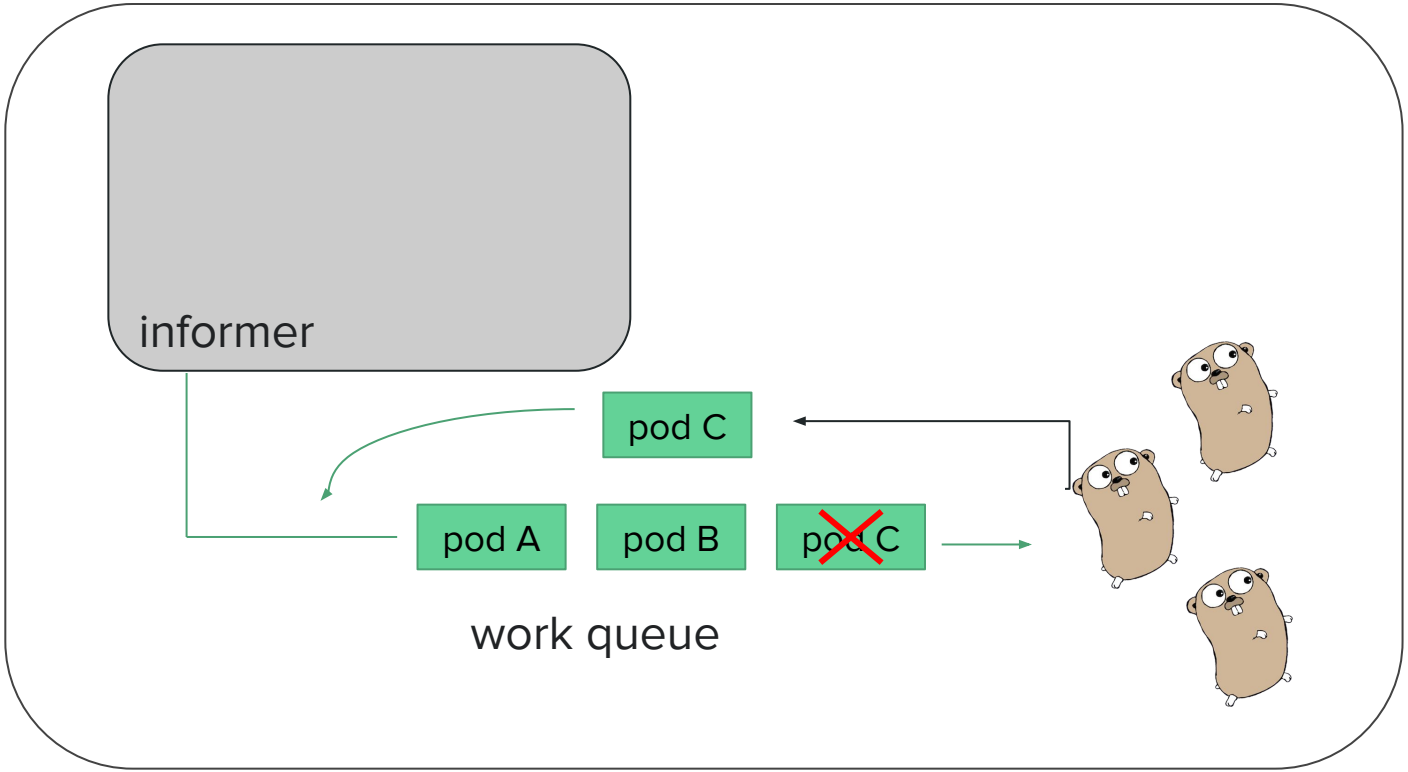
Work queues



Work queues



Work queues



Custom Resources Definitions

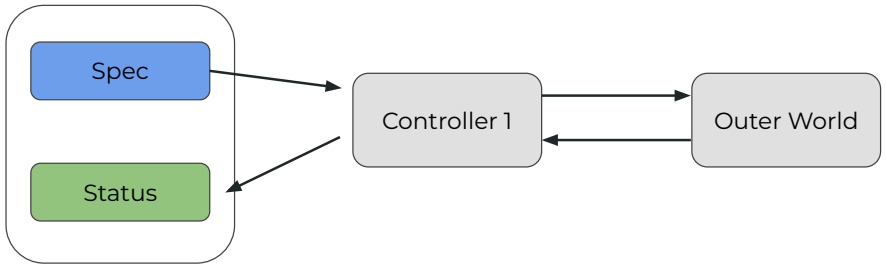
- Kubernetes types describing user defined kubernetes types
- Can be handled by controllers
- Represent our cloud-native API

Custom Resources Definitions

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
    - ct
```

Spec is the desired state, status the actual state

```yaml
apiVersion: example.io/v1alpha1
kind: Thermostat
metadata:
  name: high
  namespace: default
spec:
  temperature: 100
status:
  state: "on"
  temperature: 75
```

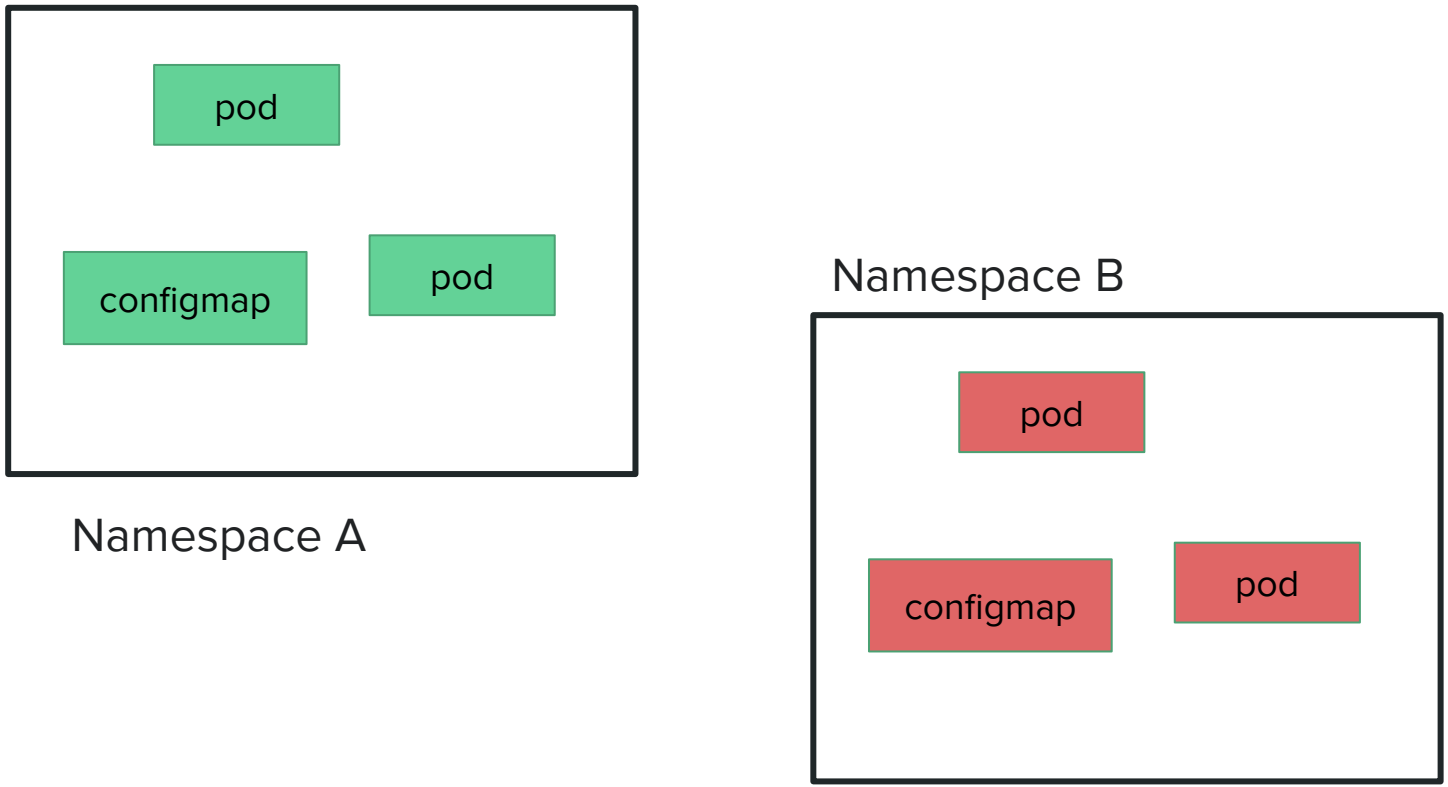


Namespaces

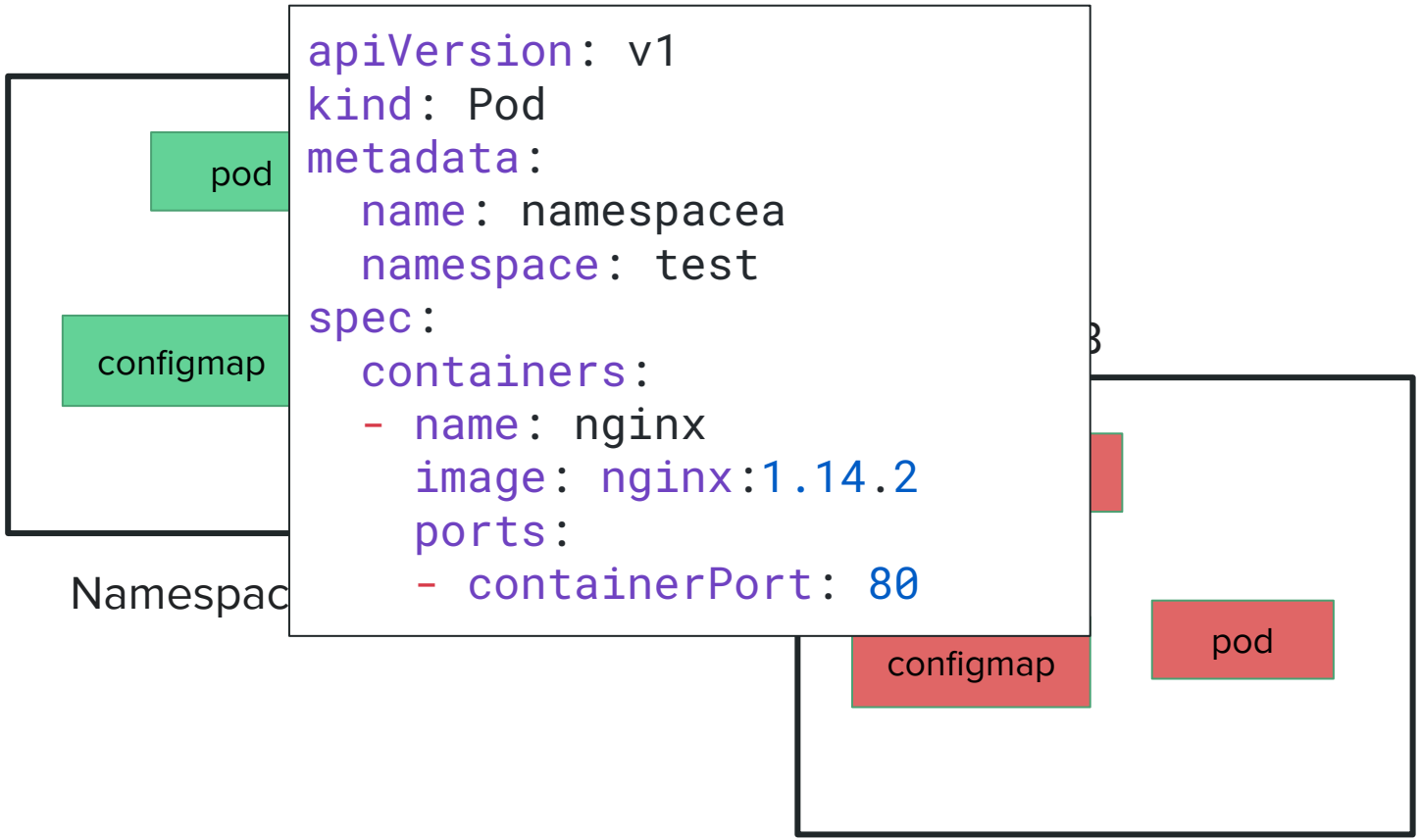
Namespaces

- Logical partitions across Kubernetes clusters
- Isolation of resources:
  - Pods
  - Configmaps
  - CRs
- Used for:
  - Multitenancy
  - Different environments

Namespaces


pod
configmap
pod
Namespace A
Namespace B
pod
configmap
pod

Namespaces


pod
configmap
Namespac
apiVersion: v1
kind: Pod
metadata:
name: namespacea
namespace: test
spec:
containers:
- name: nginx
image: nginx:1.14.2
ports:
- containerPort: 80
configmap
pod

The permission model

RBAC?

Role

Based

Access

Control

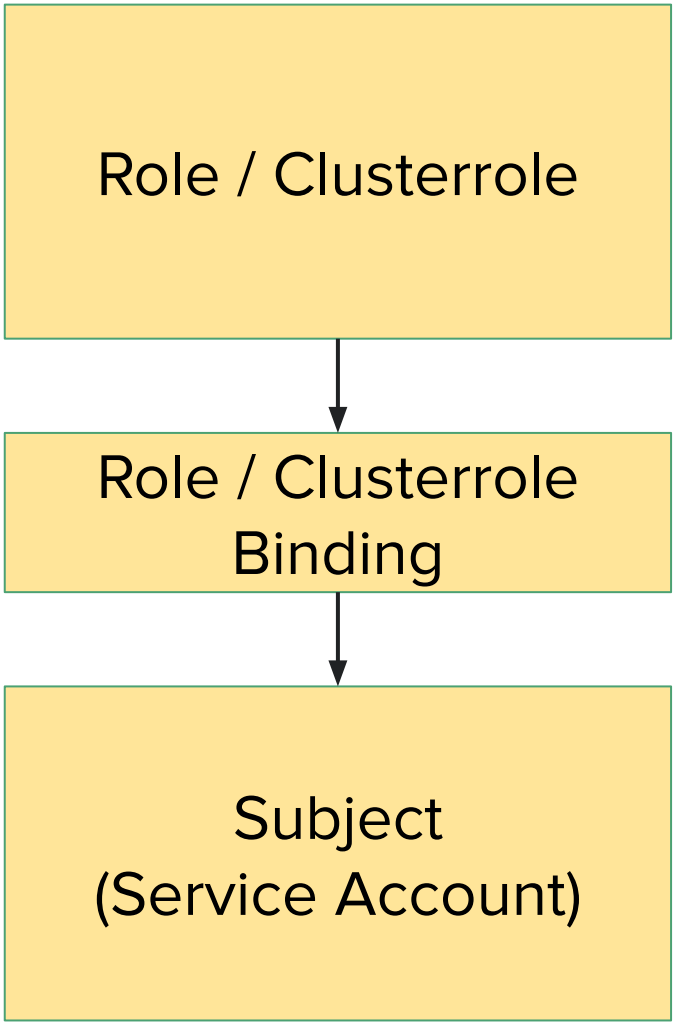
Role / Clusterrole

```
- apiGroups:
["example.golab.io"]
  resources:
    - thermostats
  verbs:
    - get
    - list
    - watch
```

Role / Clusterrole

Subject
(Service Account)

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: golabcontroller
```




```yaml
apiVersion:
rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: golab:controller
roleRef:
  apiGroup:
rbac.authorization.k8s.io
  kind: ClusterRole
  name: golab:controller
subjects:
  - kind: ServiceAccount
    name: golabcontroller
```

Role vs ClusterRole

- Cluster role provides per cluster permissions
- Role provides per namespace permissions

Too Many Yamls!

Kubebuilder

- A framework for building Kubernetes APIs using Go.
- Simplifies creation of Custom Resource Definitions (CRDs) and controllers.
- Part of the official Kubernetes SIG API Machinery.
- Leverages Controller Runtime

kubebuilder.io

What Kubebuilder does

- Generates the project scaffolding
- Generates the yamls for the CRDs
- Generates the yamls for the permissions
- Generates the boilerplate code for the reconciliation loop

kubebuilder.io

Project generation

Kubebuilder

```
kubebuilder init --domain golab.io --repo golab.io/kubedredger
```

Kubebuilder

```
kubebuilder init --domain golab.io --repo golab.io/kubedredger
```

```
cmd
config
Dockerfile
go.mod
go.sum
hack
Makefile
PROJECT
README.md
test
```

Makefile rules

- make docker-build
- make build
- make lint
- make test
- make setup-e2e
- make e2e
- ....

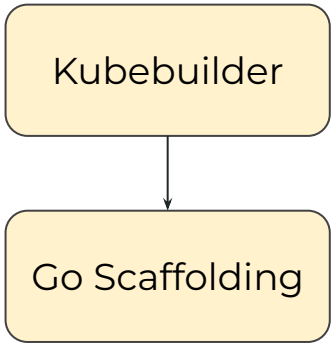
API

The workflow

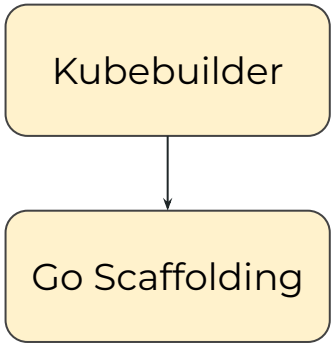
Kubebuilder

```
kubebuilder create api --group workshop --version v1alpha1 \
--kind Configuration
```

The workflow



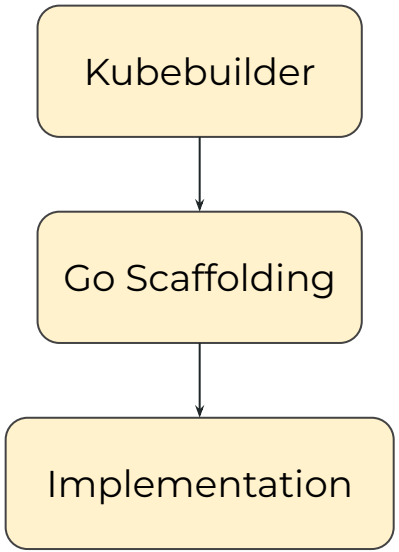
The workflow




```
api
└── v1alpha1
    ├── configuration_types.go
    ├── groupversion_info.go
    └── zz_generated.deepcopy.go
```

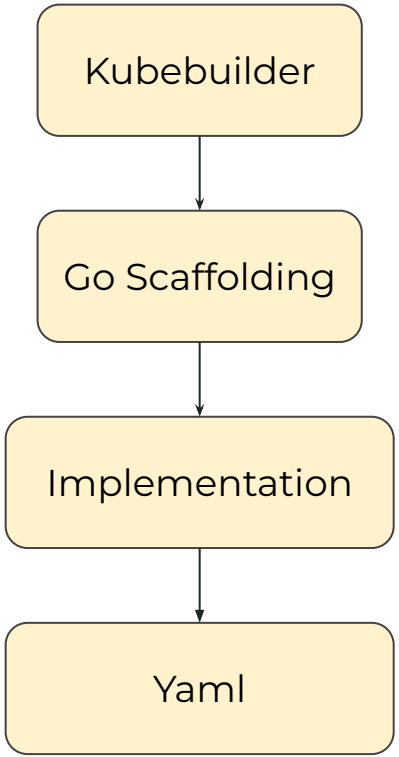
```go
// ConfigurationSpec defines the desired state of Configuration
type ConfigurationSpec struct {
        // INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
        Foo *string `json:"foo,omitempty"`
}
```

The workflow



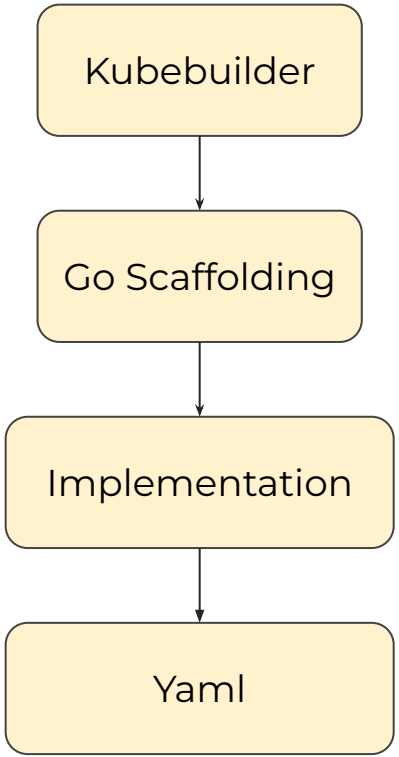

```go
// ConfigurationSpec defines the desired state of Configuration

type ConfigurationSpec struct {
  // Content is the content to be written to the file
  Content string `json:"content"`
  // Create indicates whether to create the file if it does not exist
  Create bool `json:"create,omitempty"`
  // +optional
  Permission *uint32 `json:"permission,omitempty"`
}
```

The workflow




```
make manifests
```

The workflow



`make manifests`

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    controller-gen.kubebuilder.io/version: v0.18.0
  name: configurations.workshop.golab.io
spec:
  group: workshop.golab.io
  names:
    kind: Configuration
    listKind: ConfigurationList
    plural: configurations
    singular: configuration
  scope: Namespaced
  versions:
  - name: v1alpha1
    schema:
      openAPIV3Schema:
        description: Configuration is the Schema for the
configurations API
```

API Properties

Kubebuilder

```
// Foo is a foo field
// +kubebuilder:validation:Minimum=1
// +kubebuilder:default:=32
// +optional
Foo *int32 `json:"aggregationLength,omitempty"`
```

Kubebuilder

```
// Foo is a foo field
// +kubebuilder:validation:Minimum=1
// +kubebuilder:default:=32
// +optional
Foo *int32 `json:"aggregationLength,omitempty"`
```

Kubebuilder

```
// Foo is a foo field
// +kubebuilder:validation:Minimum=1
// +kubebuilder:default:=32
// +optional
Foo *int32 `json:"aggregationLength,omitempty"`
```

Kubebuilder

```
// Foo is a foo field
// +kubebuilder:validation:Minimum=1
// +kubebuilder:default:=32
// +optional
Foo *int32 `json:"aggregationLength,omitempty"`
```

Kubebuilder

```
// Foo is a foo field
// +kubebuilder:validation:Minimum=1
// +kubebuilder:default:=32
// +optional
Foo *int32 `json:"aggregationLength,omitempty"`
```

Print columns

```
//
+kubebuilder:printcolumn:name="Path",type="string",JSONPath=".spec.path",description=
"Path of the file"
//
+kubebuilder:printcolumn:name="Exists",type="boolean",JSONPath=".status.fileExists",d
escription="Tells if the file exists"
//
+kubebuilder:printcolumn:name="LastUpdate",type="date",JSONPath=".status.lastUpdated"
,description="Last update of the file"
```

```
NAMESPACE                          NAME          PATH          EXISTS
LASTUPDATE
k8s-example-kubedredger-system     test-config   /foo.txt      true      16s
```

RBAC generation

RBAC

```
//+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/status,verbs=get;update;patch
// +kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/finalizers,verbs=update
```

```
make manifests
```

RBAC

```
//+kubebuilder:rbac:                                                    tch;create;update;patc
h;delete
// +kubebuilder:rbac                                                    ;update;patch
// +kubebuilder:rbac                                                    =update
```

```
make manif
```

```yaml
apiVersion:
rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: manager-role
rules:
- apiGroups:
  - workshop.golab.io
  resources:
  - configurations
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
```

KIND: Running a cluster on your laptop

KIND: Kubernetes in Docker

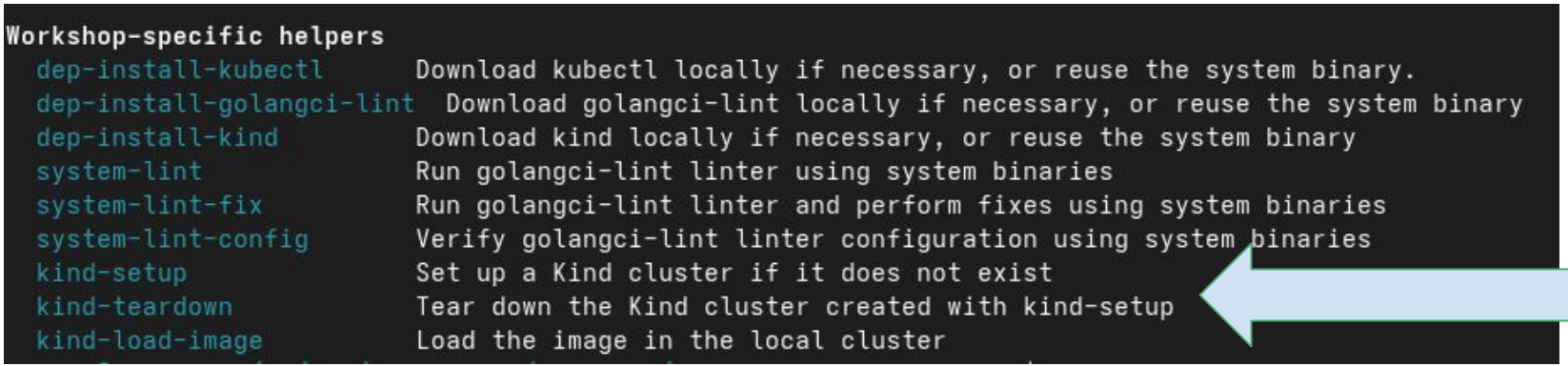
- a tool for running local Kubernetes clusters using Docker container “nodes”.
- primarily designed for testing Kubernetes itself, but may be used for local development or CI.
- you need: kind itself, kubectl, docker (or podman)

kind.sigs.k8s.io

KIND: for this workshop...

https://github.com/goworkshops/k8s-example-kubedredger

- make help - generic usage
- **make dep-install-kubectl**
- **make dep-install-kind**



KIND: for this workshop... /2

- in the repo root
- setup the kind cluster: **make kind-setup**
- teardown once done: **make kind-teardown**
- Your cluster is ready!

KIND: for this workshop...

```
$ make kind-setup
Kind cluster 'k8s-example-kubedredger-kind' already exists. Skipping creation.
$ make kind-teardown
Deleting cluster "k8s-example-kubedredger-kind" ...
Deleted nodes: ["k8s-example-kubedredger-kind-control-plane"]
$ make kind-setup
No kind clusters found.
Creating Kind cluster 'k8s-example-kubedredger-kind'...
Creating cluster "k8s-example-kubedredger-kind" ...
 ✓ Ensuring node image (kindest/node:v1.34.0) 🖼
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-k8s-example-kubedredger-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-k8s-example-kubedredger-kind

Thanks for using kind! 😊
$ make kind-teardown
Deleting cluster "k8s-example-kubedredger-kind" ...
Deleted nodes: ["k8s-example-kubedredger-kind-control-plane"]
$
```

Kubectl: accessing a kubernetes cluster

What is it: a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API. Usage examples.

If you don’t have it already:

- make kubectl # from the repo root

Kubectl: example 1: checking versions

```
$ kubectl version
Client Version: v1.34.1
Kustomize Version: v5.7.1
Server Version: v1.34.0
```

Kubectl: example 2: getting “nodes”

```
$ kubectl get nodes
NAME                                              STATUS   ROLES           AGE     VERSION
k8s-example-kubedredger-kind-control-plane   Ready    control-plane   2m34s   v1.34.0
```

need more? please check hack/kind*.yaml

Kubectl: example 3: getting all pods

```
$ kubectl get pods -A
NAMESPACE            NAME                                                         READY   STATUS    RESTARTS   AGE
kube-system          coredns-66bc5c9577-b5fp9                                     1/1     Running   0          2m37s
kube-system          coredns-66bc5c9577-q6zk9                                     1/1     Running   0          2m37s
kube-system          etcd-k8s-example-kubedredger-kind-control-plane              1/1     Running   0          2m44s
kube-system          kindnet-ksf8f                                                1/1     Running   0          2m37s
kube-system          kube-apiserver-k8s-example-kubedredger-kind-control-plane    1/1     Running   0          2m44s
kube-system          kube-controller-manager-k8s-example-kubedredger-kind-control-plane   1/1     Running   0          2m44s
kube-system          kube-proxy-vpgbr                                             1/1     Running   0          2m37s
kube-system          kube-scheduler-k8s-example-kubedredger-kind-control-plane    1/1     Running   0          2m44s
local-path-storage   local-path-provisioner-7b8c8ddbd6-8tkd4                      1/1     Running   0          2m37s

$ kubectl get pods -n local-path-storage
NAME                                     READY   STATUS    RESTARTS   AGE
local-path-provisioner-7b8c8ddbd6-8tkd4   1/1     Running   0          19m
```

Kubectl: example 4: getting a pod YAML

```
$ kubectl get pod -o yaml -n kube-system kindnet-ksf8f
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-09-28T20:17:02Z"
  generateName: kindnet-
  generation: 1
  labels:
    app: kindnet
    controller-revision-hash: 57c957c676
    k8s-app: kindnet
    pod-template-generation: "1"
    tier: node
  name: kindnet-ksf8f
...
```

Kubectl: example 5: getting container logs

```
$ kubectl logs -n kube-system kindnet-ksf8f
I0928 20:17:04.161842       1 main.go:390] probe TCP address
k8s-example-kubedredger-kind-control-plane:6443
I0928 20:17:04.163231       1 main.go:109] connected to apiserver:
https://k8s-example-kubedredger-kind-control-plane:6443
I0928 20:17:04.163386       1 main.go:139] hostIP = 172.18.0.2
podIP = 172.18.0.2
I0928 20:17:04.163533       1 main.go:148] setting mtu 1500 for CNI
I0928 20:17:04.163545       1 main.go:178] kindnetd IP family: "ipv4"
I0928 20:17:04.163555       1 main.go:182] noMask IPv4 subnets: [10.244.0.0/16]
...
```

Kubectl: (many) more examples

kubectl quick reference

Running in a cluster

How to run my container?

1. create container image
2. upload container image to a registry
3. send manifests to the cluster referring the image
4. success!

How to run my container? for this workshop

1. make container-image
2. make kind-load-image
3. make deploy
4. success!

Make container-image

Create the container image based on the Dockerfile

calls docker build with the correct parameters

Make kind-load-image

regenerate partial manifests from annotations (implied dependency)

create manifests from the template

sends the manifests to the (kind) cluster

Make deploy

Load the controller container image in the kind cluster directly in its data store

- avoids the need to push into a container registry
- requires **imagePullPolicy: ifNotPresent** in the manifests referring the image
- otherwise kubernetes will try to use a registry

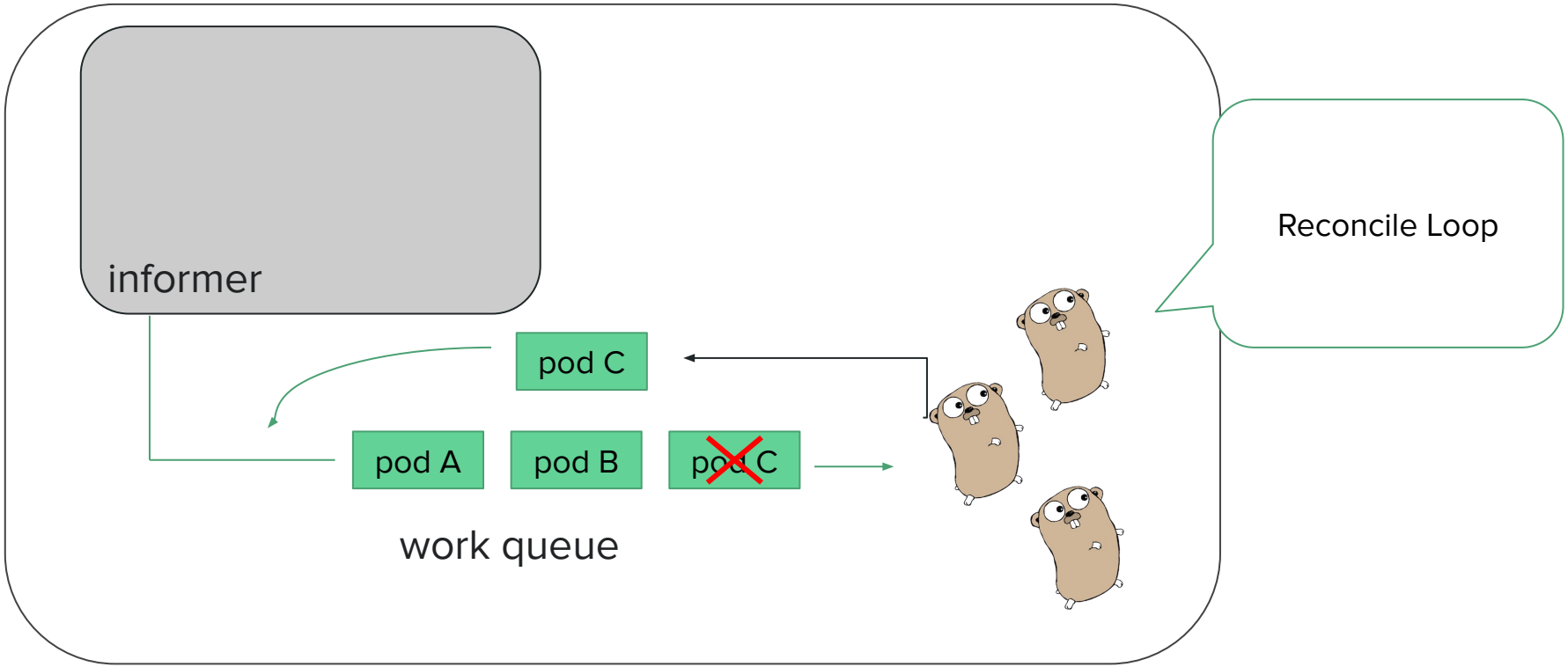
How to run my container? for this workshop

Optional but recommended activity: try to look the targets in the Makefile and look at the commands they run

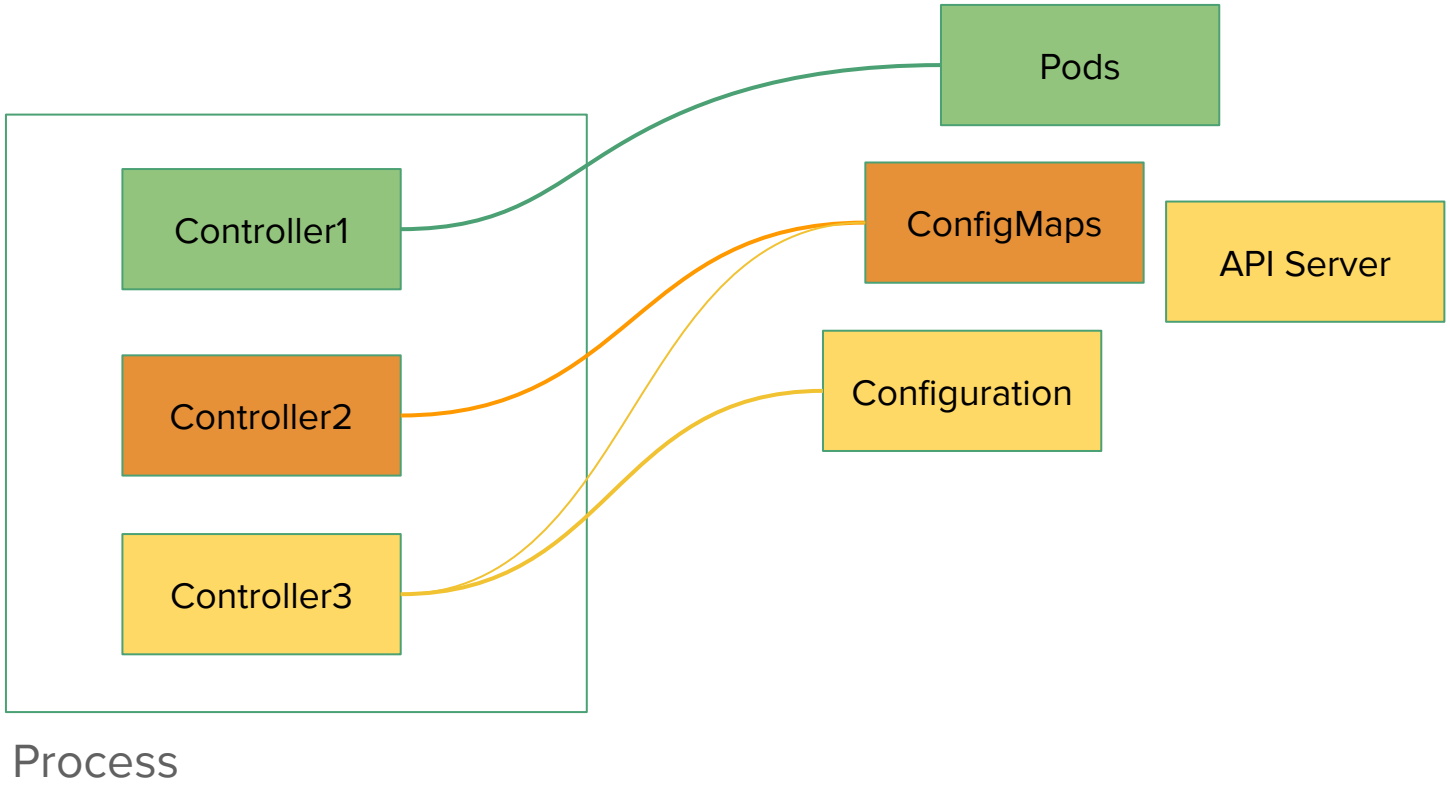
1. make container-image
2. make kind-load-image
3. make deploy

Writing the reconcile loop

What's a controller?



What's a controller?



Kubebuilder uses controller-runtime

Controller runtime is a convenient wrapper of clients, caches, informers, watches

Building Blocks

- Reconcile loop
- Cache / Client to access resources

Automatic scaffolding generation

```
kubebuilder create api --group workshop --version v1alpha1 --kind
Configuration
INFO Create Resource [y/n]
y
INFO Create Controller [y/n]
y
INFO Writing kustomize manifests for you to edit...
INFO Writing scaffold for you to edit...
INFO api/v1alpha1/configuration_types.go
INFO api/v1alpha1/groupversion_info.go
INFO internal/controller/suite_test.go
INFO internal/controller/configuration_controller.go
INFO internal/controller/configuration_controller_test.go
```

What we get

```
type ConfigurationReconciler struct {
        client.Client
        Scheme *runtime.Scheme
}
```

What we get

```
type ConfigurationReconciler struct {
        client.Client
        Scheme *runtime.Scheme
}
```

Our controller type

- Implements the reconcile loop
- Hosts fields needed to implement the business logic and to tie Kubernetes to the real world

```
//
+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations,verbs=ge
t;list;watch;create;update;patch;delete
//
+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/status,v
erbs=get;update;patch
//
+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/finalize
rs,verbs=update

func (r *ConfigurationReconciler) Reconcile(ctx context.Context, req
ctrl.Request) (ctrl.Result, error) {
        _ = logf.FromContext(ctx)

        // TODO(user): your logic here

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

```
//
+kubebuilder:rbac:groups=workshop.go
t;list;watch;create;update;patch;del
//
+kubebuilder:rbac:groups=workshop.go
erbs=get;update;patch
//
+kubebuilder:rbac:groups=workshop.go
rs,verbs=update

func (r *ConfigurationReconciler) Reconcile(ctx context.Context, req
ctrl.Request) (ctrl.Result, error) {
        _ = logf.FromContext(ctx)

        // TODO(user): your logic here

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

The reconcile loop. Gets triggered every time a resource we are monitoring is changed

```
// SetupWithManager sets up the controller with the Manager.
func (r *ConfigurationReconciler) SetupWithManager(mgr ctrl.Manager)
error {
        return ctrl.NewControllerManagedBy(mgr).
                For(&workshopvalpha.Configuration{}).
                Named("configuration").
                Complete(r)
}
```

Wiring to the controller manager.
We declare the objects we are interested into

```
// SetupWithManager sets up the controller with the Manager.
func (r *ConfigurationReconciler) SetupWithManager(mgr ctrl.Manager)
error {
        return ctrl.NewControllerManagedBy(mgr).
                For(&workshopvalpha.Configuration{}).
                Named("configuration").
                Complete(r)
}
```

Let’s focus on Reconcile

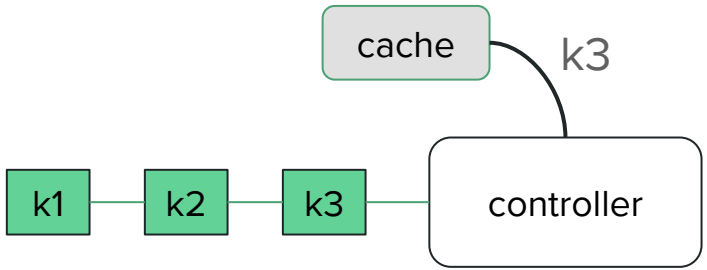
```
func (r *ConfigurationReconciler) Reconcile(ctx context.Context,
                                            req ctrl.Request) (ctrl.Result, error) {
        lh := logf.FromContext(ctx)

        conf := workshopv1alpha1.Configuration{}
        err := r.Get(ctx, req.NamespacedName, &conf)

        if apierrors.IsNotFound(err) {
                // handle deletion
        }
```

```
type NamespacedName struct {
    Namespace string
    Name      string
}
```

```go
func (r *ConfigurationReconciler) Reco...le(ctx context.Context,
                                        req ctrl.Request) (ctrl.Result, error) {
    lh := logf.FromContext(ctx)

    conf := workshopv1alpha1.Configuration{}
    err := r.Get(ctx, req.NamespacedName, &conf)

    if apierrors.IsNotFound(err) {
        // handle deletion
    }
```

Req contains the key of the object, not the object itself

- To access the object, it must be fetched using the client
- No object means the object was cancelled
- No old values and new values!



The return values

```
// If the returned error is non-nil, the Result is ignored and
// the request will be requeued using exponential backoff.
// The only exception is if the error is a TerminalError
// in which case no requeuing happens.
//
// If the error is nil and the returned Result has a non-zero
// result.RequeueAfter, the request will be requeued after
// the specified duration.

Reconcile(context.Context, request) (Result, error)
```

The return values

```
// If the returned error is non-nil, the Result is ignored and
// the request will be requeued using exponential backoff.
// The only exception is if the error is a TerminalError
// in which case no requeuing happens.
//
// If the error is nil and the returned Result has a non-zero
// result.RequeueAfter, the request will be requeued after
// the specified duration.

Reconcile(context.Context, request) (Result, error)
```

The return values

```
// If the returned error is non-nil, the Result is ignored and
// the request will be requeued using exponential backoff.
// The only exception is if the error is a TerminalError
// in which case no requeuing happens.
//
// If the error is nil and the returned Result has a non-zero
// result.RequeueAfter, the request will be requeued after
// the specified duration.

Reconcile(context.Context, request) (Result, error)
```

The return values

```
// If the returned error is non-nil, the Result is ignored and
// the request will be requeued using exponential backoff.
// The only exception is if the error is a TerminalError
// in which case no requeuing happens.
//
// If the error is nil and the returned Result has a non-zero
//  result.RequeueAfter, the request will be requeued after
// the specified duration.

Reconcile(context.Context, request) (Result, error)
```

Anatomy of a good controller

Anatomy of a good controller

- business logic has a clear boundary, with a its own API
- the reconciliation logic is IDEMPOTENT
- any transient error must be sent back to the queue
- any non transient error must be logged and Reconcile must return success

```
type ConfigurationReconciler struct {
        client.Client
        Scheme   *runtime.Scheme
        ConfMgr *configfile.Manager
}
```

The business logic we want to implement

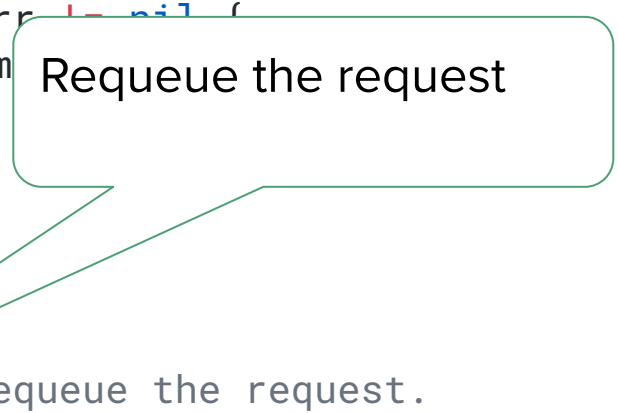
```
err := r.Get(ctx, req.NamespacedName, &conf)
if apierrors.IsNotFound(err) {
        if err := r.ConfMgr.Delete(); err != nil {
                return ctrl.Result{}, fmt.Errorf("failed to delete the
configuration: %w", err)
        }
        return ctrl.Result{}, nil
}

if err != nil {
        // Error reading the object - requeue the request.
        return ctrl.Result{}, err
}
```

```
err := r.Get(ctx, req.NamespacedName, &conf)
if apierrors.IsNotFound(err) {
        if err := r.ConfMgr.Delete(); err != nil {
                return ctrl.Result{}, 
configuration: %w", err)
        }
        return ctrl.Result{}, nil
}

if err != nil {
        // Error reading the object - requeue the request.
        return ctrl.Result{}, err
}
```

Handle the deletion

the

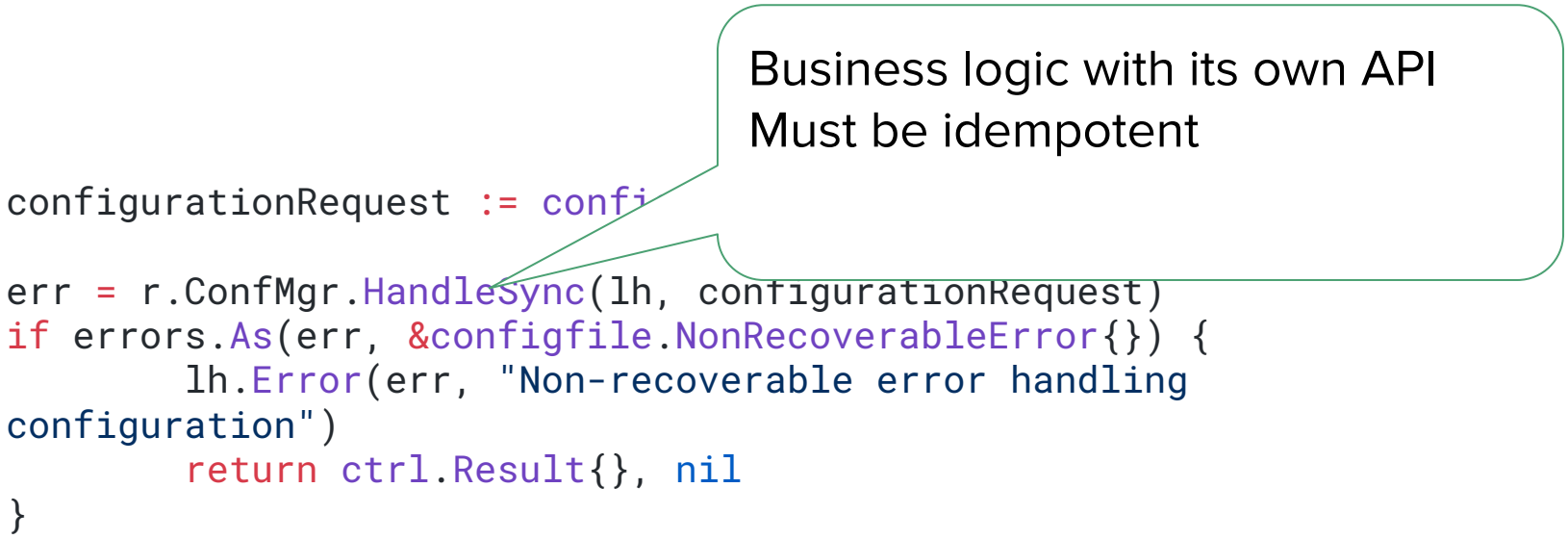
```
err := r.Get(ctx, req.NamespacedName, &conf)
if apierrors.IsNotFound(err) {
        if err := r.ConfMgr.Delete(); err != nil {
                return ctrl.Result{}, fm                                    the
configuration: %w", err)
        }
        return ctrl.Result{}, nil
}

if err != nil {
        // Error reading the object - requeue the request.
        return ctrl.Result{}, err
}
```



```
configurationRequest := configurationRequestFromSpec(conf.Spec)

err = r.ConfMgr.HandleSync(lh, configurationRequest)
if errors.As(err, &configfile.NonRecoverableError{}) {
        lh.Error(err, "Non-recoverable error handling
configuration")
        return ctrl.Result{}, nil
}
```

Translation from the kubernetes API to internal

```
configurationRequest := configurationRequestFromSpec(conf.Spec)

err = r.ConfMgr.HandleSync(lh, configurationRequest)
if errors.As(err, &configfile.NonRecoverableError{}) {
        lh.Error(err, "Non-recoverable error handling
configuration")
        return ctrl.Result{}, nil
}
```


Business logic with its own API
Must be idempotent
configurationRequest := confi
err = r.ConfMgr.HandleSync(lh, configurationRequest)
if errors.As(err, &configfile.NonRecoverableError{}) {
lh.Error(err, "Non-recoverable error handling configuration")
return ctrl.Result{}, nil
}

Notes

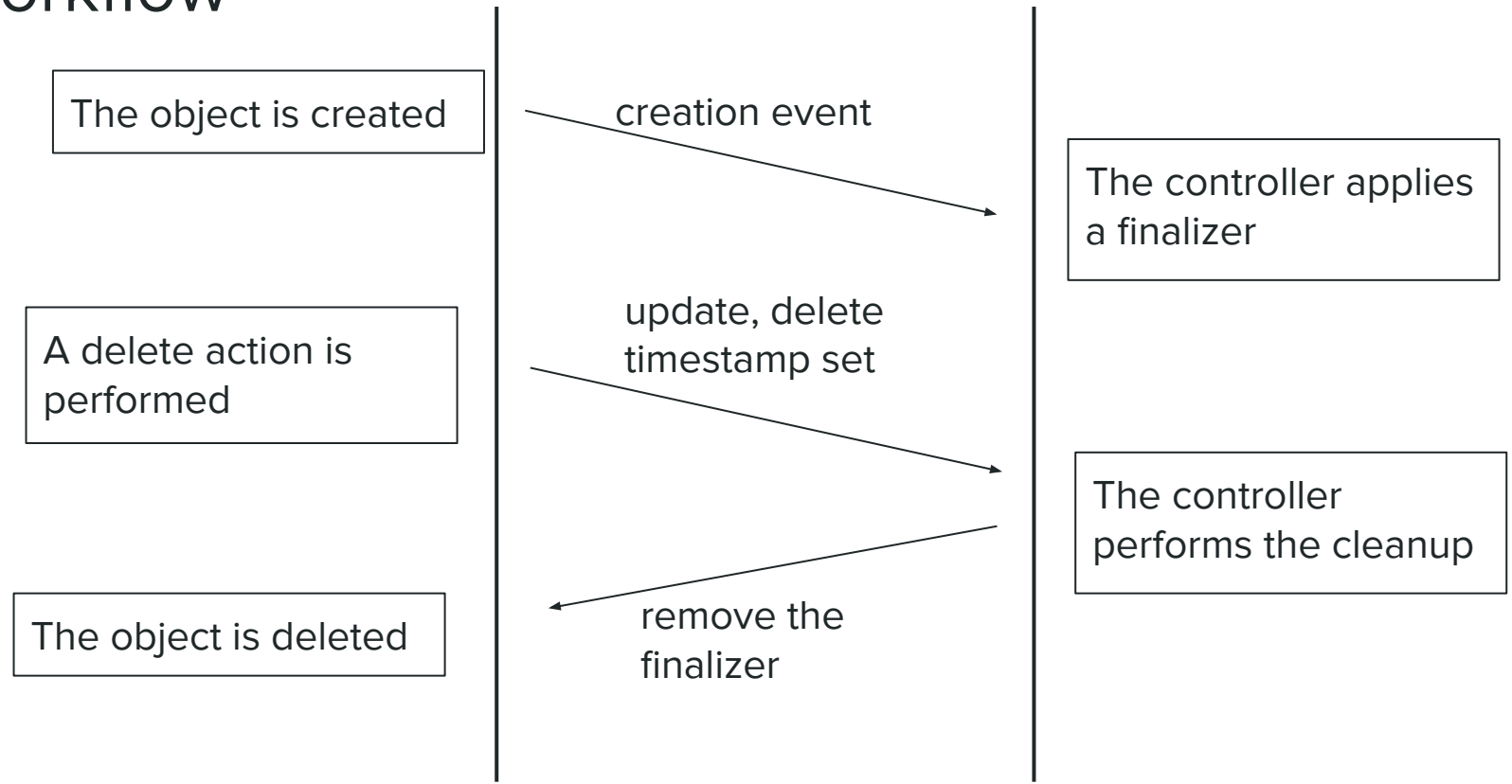
- the cache gets warmed up before reconcile is called
- when we get an object using Get or List, we access the local cache and not the latest data on the api server
- because of how the informer works, the cache is updated before the handler is called

Takeaways

- we don’t know how many times our reconcile is getting called with the same data
- non-k8s business logic allows us to test it in isolation
- the business logic must be idempotent
- the reward is a bullet proof controller

Finalizers

The workflow



Finalizers are pre-delete hooks

- An object does not get removed as long as it has a finalizer applied
- This gives time to perform housekeeping asynchronously
- Also, it allow us to use the whole object before it gets deleted

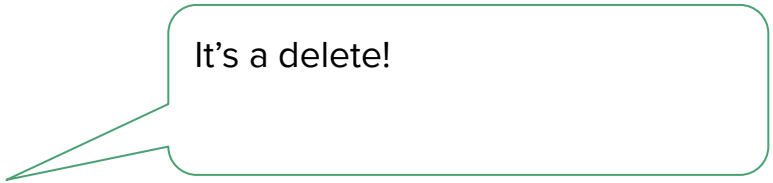
Finalizers require specific permissions

```
//+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/status,verbs=get;update;patch
// +kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/finalizers,verbs=update
```

In the reconcile loop

```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalizer(conf, Finalizer)
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result{}, err
        }
}


// RECONCILE LOGIC
```

In the reconcile loop




```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalizer(conf, Finalizer)
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result{}, err
        }
}


// RECONCILE LOGIC
```

In the reconcile loop

```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalizer(conf, Finalizer)
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result{}, err
        }
}


// RECONCILE LOGIC
```

If the finalizer is set, we need to perform the delete (or the delete is in progress)

In the reconcile loop

```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalize
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result{}, err
        }
}


// RECONCILE LOGIC
```

If not, the object is just being deleted, we can exit

In the reconcile loop

```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalizer(conf, Finalizer)
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result{}, err
        }
}


// RECONCILE LOGIC
```

Add / update. We want to set the finalizer so we control the real deletion of the object

In the reconcile loop

```
if !conf.DeletionTimestamp.IsZero() {
        if controllerutil.ContainsFinalizer(conf, Finalizer) {

                // PERFORM DELETE
                controllerutil.RemoveFinalizer(conf, Finalizer)
                err = r.Update(ctx, conf)
                return ctrl.Result{}, err
        }
        return ctrl.Result{}, nil
}

if !controllerutil.ContainsFinalizer(conf, Finalizer) {
        controllerutil.AddFinalizer(conf, Finalizer)
        err = r.Update(ctx, conf)
        if err != nil {
                return ctrl.Result
        }
}


// RECONCILE LOGIC
```

Regular reconciliation logic

Exposing the status

Status Part of the spec

```
type Pod struct {
    metav1.TypeMeta
    metav1.ObjectMeta
    Spec   PodSpec
    Status PodStatus
}
```

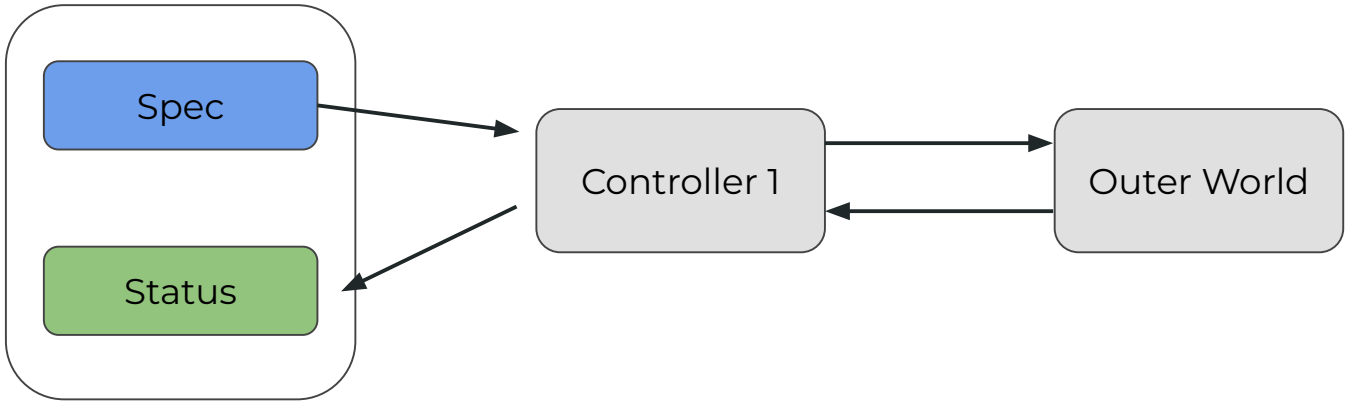
```
type MyObject struct {
    metav1.TypeMeta
    metav1.ObjectMeta
    Spec   MyObjectSpec
    Status MyObjectStatus
}
```

Provide feedback to the users

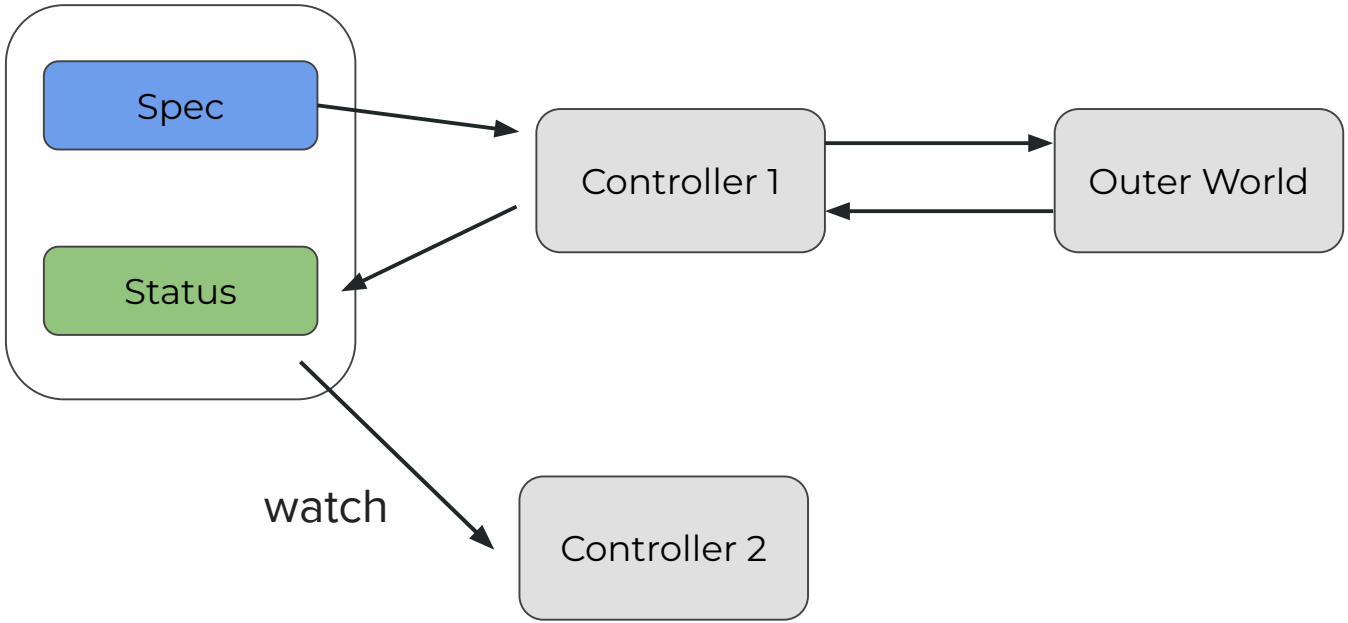
```
kubectl get pods
NAME                READY     STATUS              RESTARTS   AGE
pod0                1/1       Running             0          1h
pod1                0/1       CrashLoopBackOff    8          16m
```

The status is machine friendly

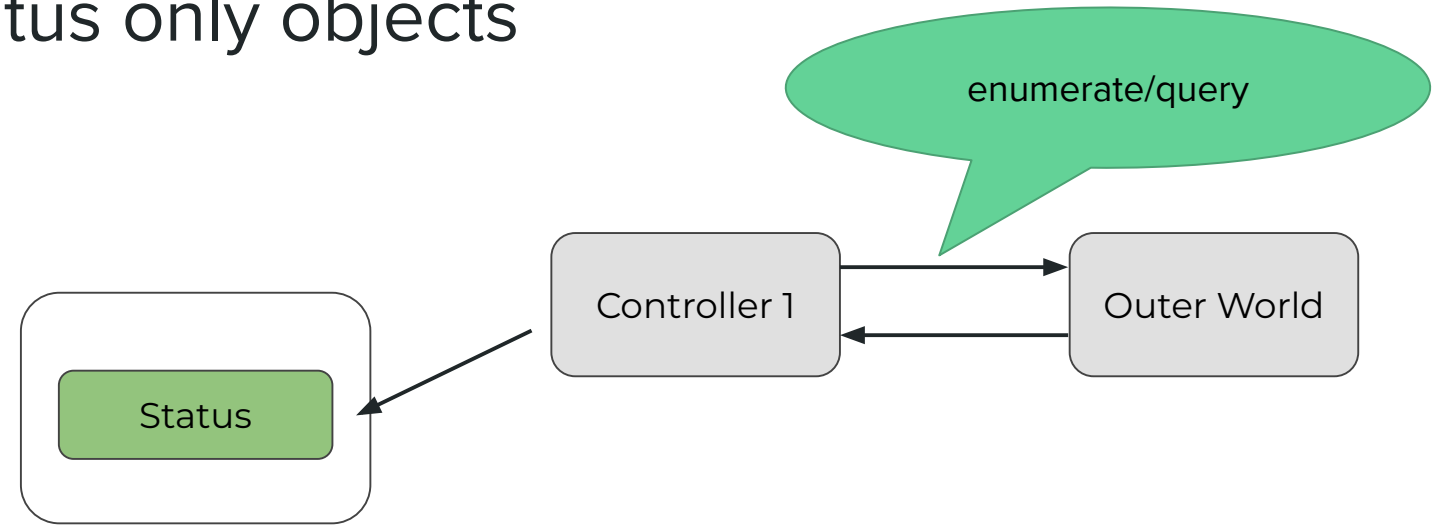
Feed other controllers



Feed other controllers



Status only objects



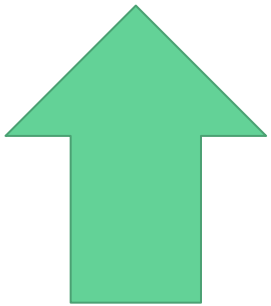
use case: enumerate resource and expose them using a kubernetes-native object

The status is just another part of the CRD

```
// ConfigurationStatus defines the observed state of Configuration.
type ConfigurationStatus struct {
  // LastUpdated is the last time the configuration was updated
  LastUpdated metav1.Time `json:"lastUpdated"`

  // Content is the current content of the file at the specified path
  Content string `json:"content,omitempty"`

  // FileExists indicates whether the file exists at the specified path
  FileExists bool `json:"fileExists,omitempty"`
  // The status of each condition is one of True, False, or Unknown.
  // +listType=map
  // +listMapKey=type
  // +optional
  Conditions []metav1.Condition `json:"conditions,omitempty"`
}
```

The status is just another part of the CRD

```
// ConfigurationStatus defines the observed state of Configuration.
type ConfigurationStatus struct {
  ...

  // The status of each condition is one of True, False, or Unknown.
  // +listType=map
  // +listMapKey=type
  // +optional
  Conditions []metav1.Condition `json:"conditions,omitempty"`
}
```



What conditions are?

It’s a convention

It’s a very good and very helpful convention we should strive to adhere to

Why conditions?

The definition of status is object-dependent

Conditions are object-independent.

Conditions provide standard, generic, machine readable status information. Great for interoperability.

Conditions

Each condition has:

- type (e.g., Available, Progressing, Degraded)
- status (True, False, Unknown)
- last transition time
- reason (for machines) & message (for humans)

The status requires a different permission

```
//
+kubebuilder:rbac:groups=workshop.golab.io,resources=configurations/
status,verbs=get;update;patch
```

```
- apiGroups:
  - workshop.golab.io
  resources:
  - configurations/status
  verbs:
  - get
  - patch
  - update
```

A simple implementation

```
oldStatus := conf.Status.DeepCopy()

// use the conf object

confManagerStatus := r.ConfMgr.Status()
conf.Status = statusFromConfManagerStatus(conf.Spec, confManagerStatus, err)

if !statusesAreEqual(oldStatus, &conf.Status) {
        updErr := r.Client.Status().Update(ctx, &conf)
        if updErr != nil {
                lh.Error(updErr, "Failed to update configuration status")
                return ctrl.Result{}, fmt.Errorf("could not update status for object")
        }
}
```

In our reconcile loop

Note the different call

```
oldStatus := conf.Status.DeepCopy()

// use the conf object

confManagerStatus := r.ConfMgr.Status()
conf.Status = statusFromConfManagerStatus(conf.Spec, confManagerStatus, err)

if !statusesAreEqual(oldStatus, &conf.Status) {
        updErr := r.Client.Status().Update(ctx, &conf)
        if updErr != nil {
                lh.Error(updErr, "Failed to update configuration status")
                return ctrl.Result{}, fmt.Errorf("could not update status for object")
        }
}
```

In our reconcile loop

```
oldStatus := conf.Status.DeepCopy()

// use the conf object

confManagerStatus := r.ConfMgr.Status()
conf.Status = statusFromConfManagerStatus(conf.Spec, confManage       , err)

if !statusesAreEqual(oldStatus, &conf.Status) {
        updErr := r.Client.Status().Update(ctx, &conf
        if updErr != nil {
                lh.Error(updErr, "Failed to update configuration status")
                return ctrl.Result{}, fmt.Errorf("could not update status for object")
        }
}
```

If the update fails, we retry

Testing a controller

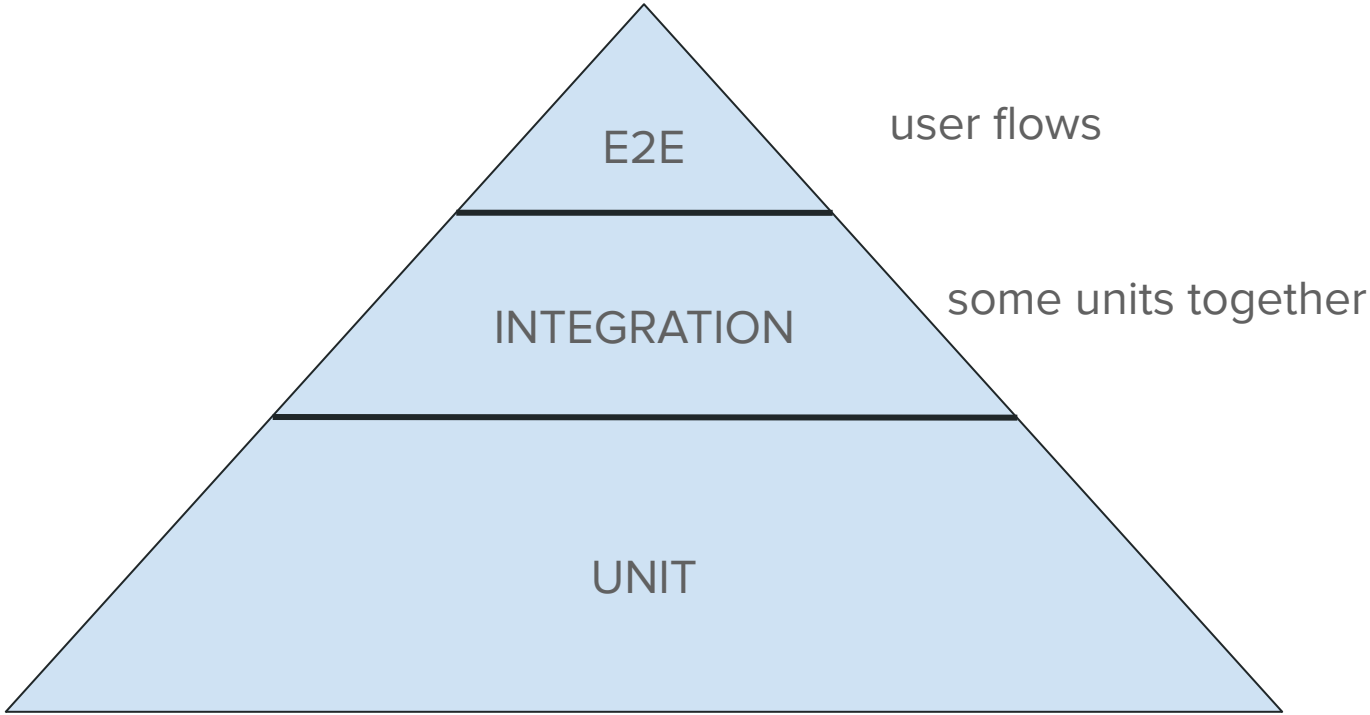
Makefile rules (reprised)

- make docker-build
- make build
- make lint
- **make test**
- **make test-e2e**
- ....

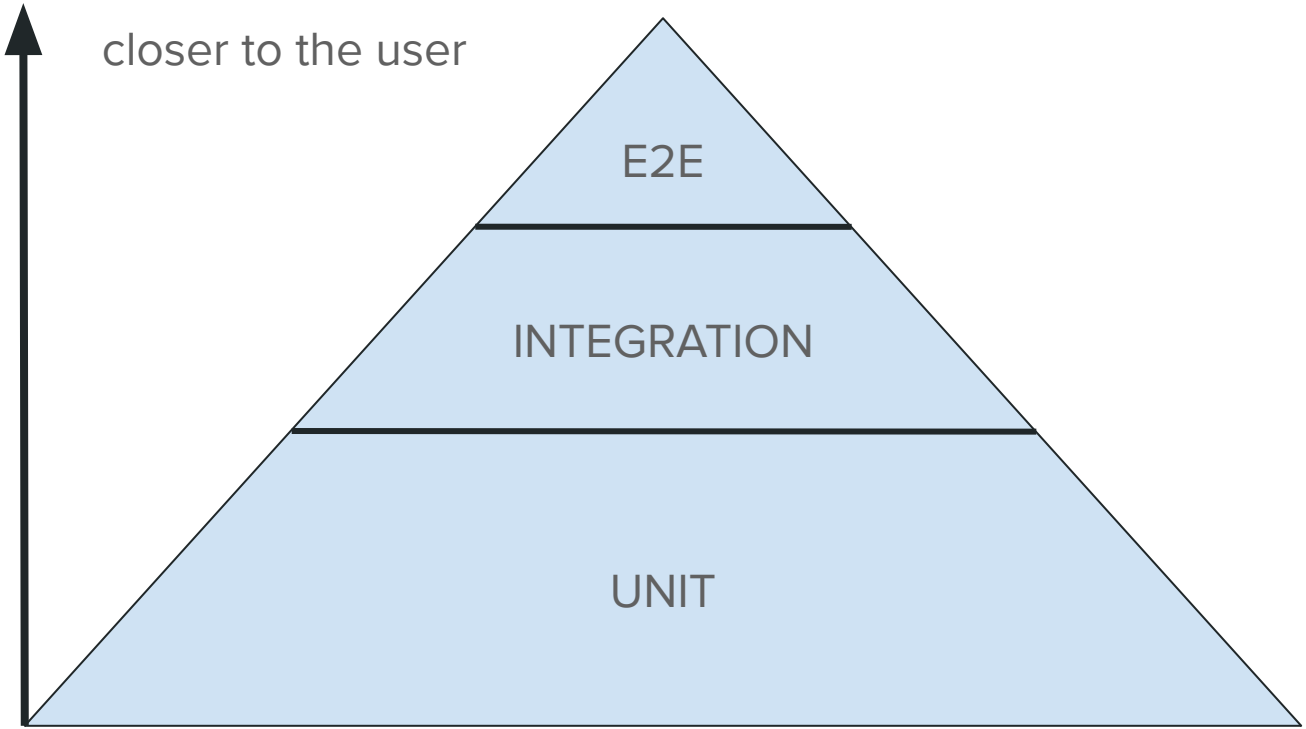
Make test

- runs the regular unit test we know and love
  - the example project intentionally only uses the stdlib testing package
- also run the controller (semi)integration tests
- e2e (end to end) tests are run with an explicit target
- running a subset of tests:
  - go test ./internal/...

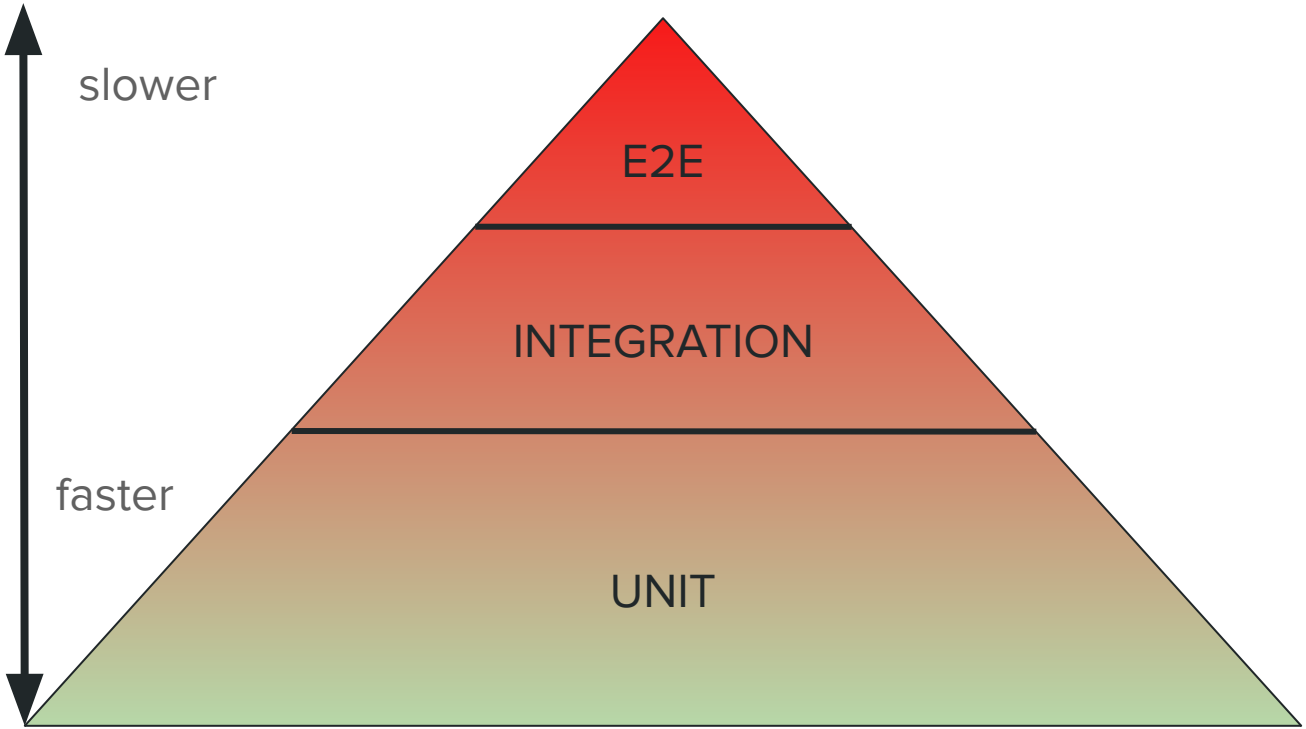
The testing pyramid



The testing pyramid



The testing pyramid: fast vs slow



unit tests: faking a go client

- The client-go package exposes a fake client implementation
- **ALMOST** native implementation look and feel
  - no server side -> no index support

```
import "sigs.k8s.io/controller-runtime/pkg/client/fake"
```

```
fake.NewClientBuilder().WithScheme(scheme.Scheme).Build()
```

unit tests: faking a go client /2

You can create objects as usual, they will end up in the fake client store

You can pre-load objects in the store (skip explicit Create in your test)

```
import "k8s.io/apimachinery/pkg/runtime"

var objects []runtime.Object

fake.NewClientBuilder().WithScheme(scheme.Scheme).WithRuntim
eObjects(objects...)
```

unit tests: faking a go client with indexer

Indexer support. E.g.

```
sel, err := fields.ParseSelector("spec.nodeName=" + nodeName)
…
podList := &corev1.PodList{}
cli.List(ctx, podList, &client.ListOptions{FieldSelector: sel})
```

unit tests: faking a go client with indexer

```
func podNodeIndexer(rawObj client.Object) []string {
    obj, ok := rawObj.(*corev1.Pod)
    if !ok {
        return []string{}
    }
    return []string{obj.Spec.NodeName}
}
```

reference (note original issue deleted)

unit tests: faking a go client with indexer

```
fake.NewClientBuilder().
    WithScheme(scheme.Scheme).
    WithIndex(&corev1.Pod{}, "spec.nodeName",podNodeIndexer).
    Build()
```

putting all together: envtest

envtest is a helper package which manages instances of etcd and the Kubernetes API server, creating a micro-cluster-like without kubelets or any other controllers.

Bare bones to test a controller, without fakes

Scaffolded by kubebuilder

Use the regular standard client

putting all together: envtest /2

```
testEnv = &envtest.Environment{
     CRDDirectoryPaths:     []string{filepath.Join("..", "..", "config", "crd", "bases")},
          ErrorIfCRDPathMissing: true,
}

cfg, err = testEnv.Start()     // cfg is defined globally

k8sClient, err = client.New(cfg, client.Options{Scheme: scheme.Scheme})

// your test code here but beware: no other controller is running!

err = testEnv.Stop()
```

Integration tests and beyond: Ginkgo and gomega

Ginkgo is a powerful testing framework for go

Gomega is a library which adds matching capabilities to ginkgo

(BeTrue, IsNil...)

Ginkgo and Gomega together provide a Domain-Specific Language (DSL) to write tests in go

The ginkgo use case: asynchronous tests

```
Eventually(X).WithTimeout(T).WithPolling(P).WithContext(ctx).Should(MATCHER)
```

Checks an assertion passes *eventually*

- tries polling every **P** time units
- until the timeout **T** expires
- optionally with a context

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      var x obj

      BeforeEach(func() {
            x = initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
```

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      BeforeEach(func() {
            initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
      })
})
```

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      BeforeEach(func() {
            initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
      })
})
```

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      BeforeEach(func() {
            initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
      })
})
```

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      BeforeEach(func() {
            initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
      })
})
```

Gingko suite zoom in

```
var _ = Describe("Some foo cases", func() {
      BeforeEach(func() {
            initialize()
      })
      Context("With some conditions", func() {
            It("should behave like this", func() {
                  Expect(foo.Something()).To(Equal(somethingElse))
            })
      })
      When("some other conditions apply", func() {
            It("should behave like that", func() {
                  Expect(bar).Should(BeTrue())
                  Expect(baz).ToNot(BeNil())
            })
      })
})
```

How ginkgo runs

ginkgo specs (= testcases) must be independent

repeating the operation multiple times produces the same result as applying it once, without any additional side effects

gingko specs runs in random order and by default in parallel

“declare in container nodes, initialize in setup nodes”

How ginkgo runs: walking the tree

Ginkgo runs in two steps: tree construction and run phase

Tree Construction:

- Ginkgo visits all container nodes, invokes their closures and constructs the spec tree.
- Ginkgo captures the relevant setup and subject node closures by visiting the tree, but **does not run them**.

How ginkgo runs: running the tree

Ginkgo runs in two steps: tree construction and run phase

Run phase:

- Ginkgo runs through each spec in the generated spec list sequentially.
- Ginkgo invokes the setup and subject nodes closures in the correct order and tracks any failed assertions, for each spec.
- Container node closures are never invoked.

Common gotchas

- All Ginkgo nodes must only appear at the top-level or within a container node.
- A subject node cannot be top level

Note: you **CAN** nest arbitrarily container nodes though!

Note: you **CAN** have multiple top-level container nodes!

Common gotchas /2

No assertion in container nodes! (ginkgo.Expect() ...)

Note: you **CAN** have any amount of assertions in a subject node!

Common gotchas /3

Do not initialize variables in container nodes

Subject nodes can mutate the values and pollute the state!

Perform initialization in setup nodes: these nodes are guaranteed to be called before every relevant subject node

Note: kinda OK for constants though - but should those be container variables?

Logging: GinkgoWriter

GinkgoWriter is a globally available io.Writer.

Aggregates everything, only emits to stdout if the test fails.

GinkgoWriter.TeeTo(writer): attach additional writers. Any data written to GinkgoWriter will immediately be sent to attached tee writers.

In verbose mode (ginkgo -v) writes to GinkgoWriter are immediately sent to stdout.

Logging: By() clause

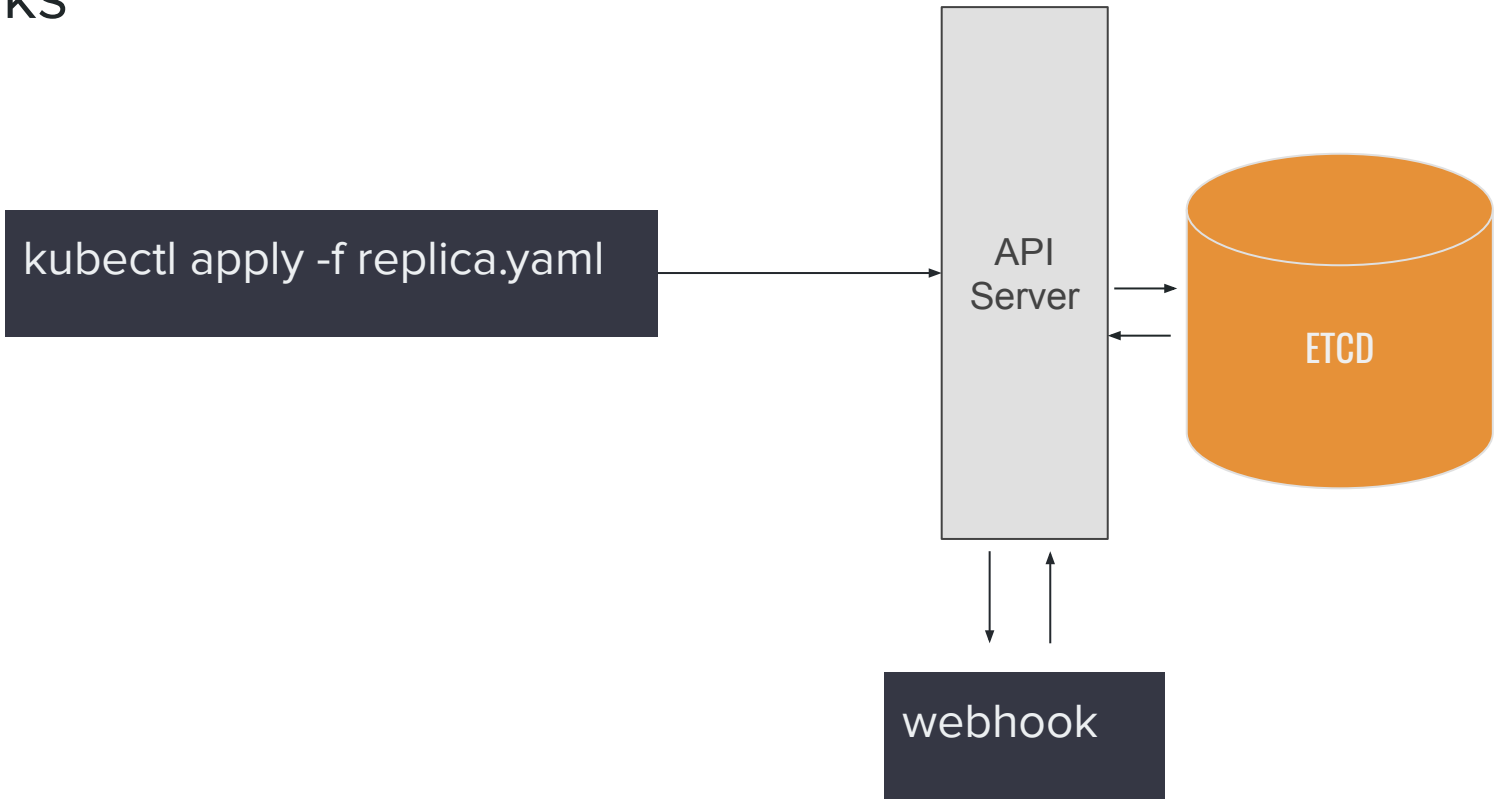
By(“my message”)

Display the messages on failure

In verbose mode, displays the steps immediately

Validating Webhooks

Webhooks



Validating Webhooks

- Bring your own validation logic
- Last resort, when the validation logic can’t be implemented with kubebuilder annotations
- Better than having to look into logs!

Validating Webhooks

```
kubebuilder create webhook --group workshop --version v1alpha1 --kind Configuration --programmatic-validation
```

```
INFO Writing kustomize manifests for you to edit...
INFO Writing scaffold for you to edit...
INFO internal/webhook/v1alpha1/configuration_webhook.go
INFO internal/webhook/v1alpha1/configuration_webhook_test.go
INFO internal/webhook/v1alpha1/webhook_suite_test.go
INFO Update dependencies
```

Validating Webhooks

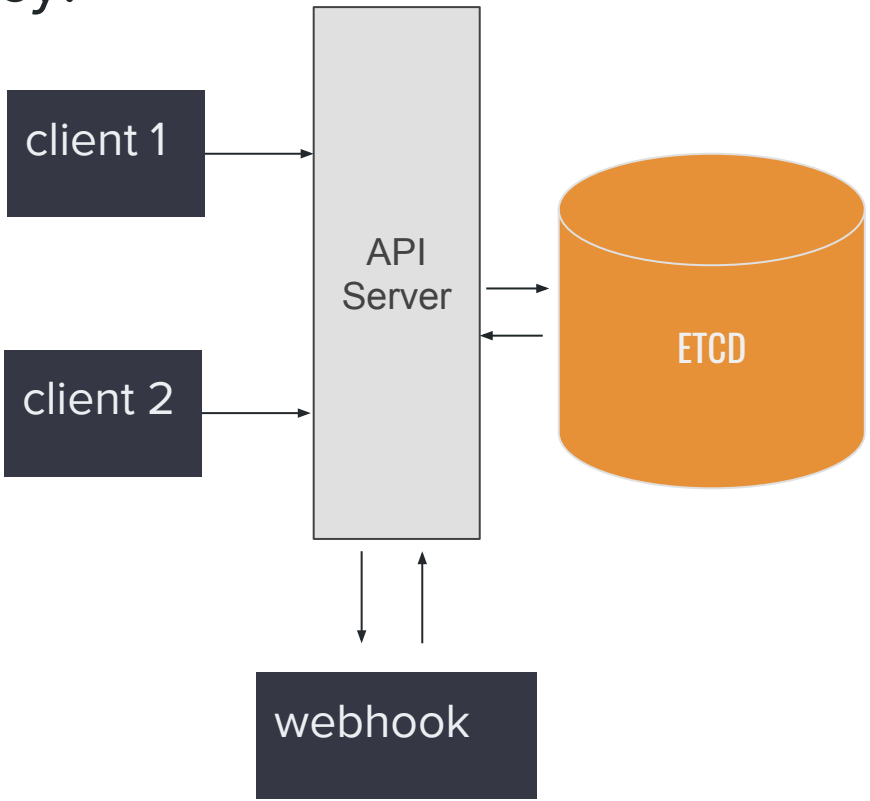
```
// ValidateCreate implements webhook.CustomValidator so a webhook will be registered for the type
Configuration.
func (v *ConfigurationCustomValidator) ValidateCreate(_ context.Context, obj runtime.Object)
(admission.Warnings, error) {
        configuration, ok := obj.(*workshopv1alpha1.Configuration)
        if !ok {
                return nil, fmt.Errorf("expected a Configuration object but got %T", obj)
        }
        configurationlog.Info("Validation for Configuration upon creation", "name",
configuration.GetName())

        // TODO(user): fill in your validation logic upon object creation.

        return nil, nil
}
```

Use webhooks when

- You need to validate multiple objects compatibility
- The validation is complicated and goes beyond the syntax
- You need to validate against the internal state

Beware of eventual consistency!

- Multiple clients might perform multiple calls against the api server
- Having a webhook does not guarantee consistency
- **Always** validate in your reconcile loop (possibly using the same logic)



Wrapping Up

Distributed systems are complicated

Failure is normal

Corner cases are hard to handle

Edge driven is tempting, level driven is more robust

Controller Runtime gives us a safe framework

Resources

- The kubebuilder book: book.kubebuilder.io
- Kubernetes custom resources: kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/

Thank you!