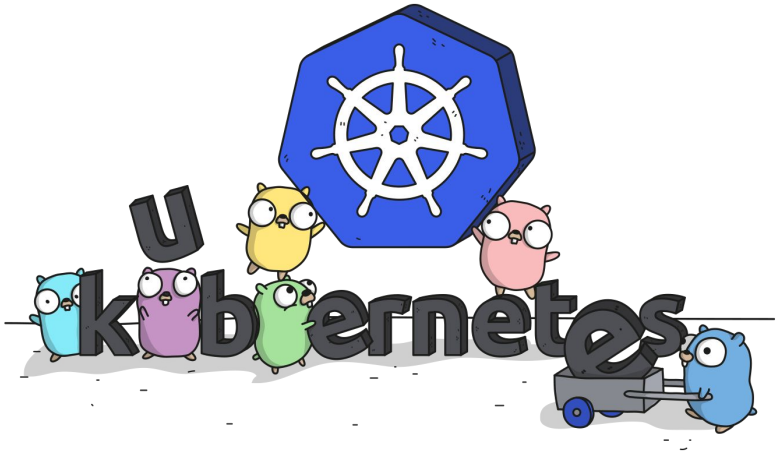
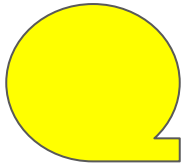


Using and extending the Kubernetes API programmatically with Go

A workshop at GopherCon UK 2018

Stefan Schimanski sttts@redhat.com
Michael Hausenblas mhausenb@redhat.com



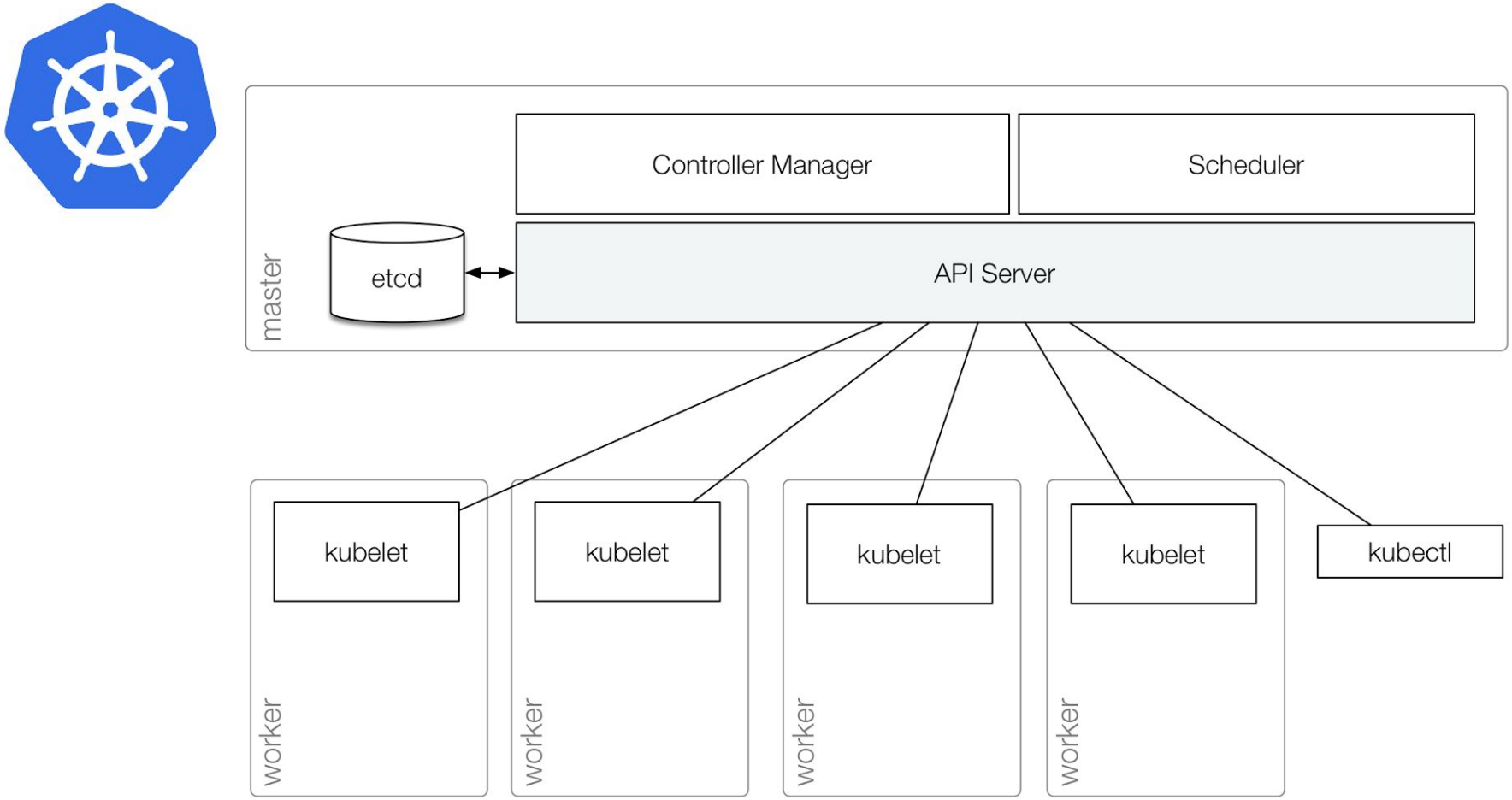
Slides

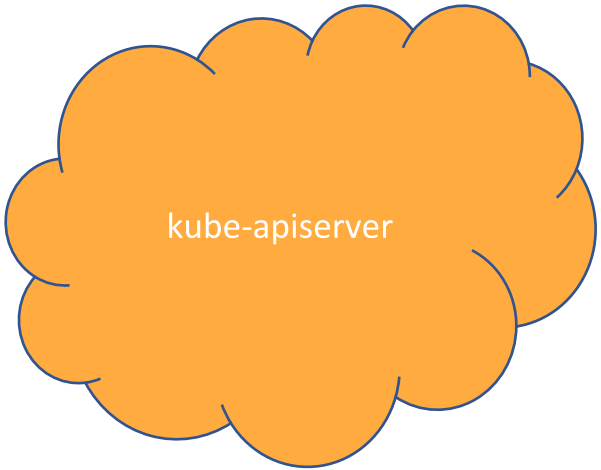
https://301.sh/2018-gopherconuk-slides

Agenda

- **API basics:** apigroups, resource vs. kind, versioning, discovery, **curl scenario**
- **Client-go basics:** clientsets, kubeconfig, **client-go scenario**, informers, **informer scenario**
- **Kubernetes objects in Golang:** ObjectMeta, TypeMeta
- **API Machinery:** Scheme, GVK, GVR, RestMapper
- **CRDs from user point of view:** CRD yaml, Validation, **CRD yaml scenario**
- **Code-Generation:** deepcopy-gen, client-gen, informer-gen, lister-gen
- **CRDs from developer point of view:** types.go, register.go, pkg/apis, code-generator, **CRD Go scenario**
- **Controllers:** controller loop, optimistic concurrency, edge-vs-level driven, **controller scenario**
- **Kubebuilder:** init, create api, **kubebuilder scenario – trainer led**
- **Operator SDK:** new operator, **operator scenario – trainer led**
- **Other topics:** deployment / in-cluster config, RBAC rules, service account, subresources, versioning, **misc scenario ("deploy your controller in the cluster")**

Kubernetes API basics


Controller Manager
Scheduler
master
etcd
API Server
kubelet
kubelet
kubelet
kubelet
kubectl
worker
worker
worker
worker



Restful
http
API

/
/version

/api
/api/v1/**pods**
/api/v1/**pods**/<name>
/api/v1/**pods**/<name>/status

/apis
/apis/batch
/apis/batch/v2alpha1
/apis/batch/v2alpha1/**jobs**
/apis/batch/v2alpha1/**cronjobs**
/apis/batch/v1
/apis/batch/v1/**jobs**

User:

$ kubectl create -f foo.yaml

Node:

proxy

kubelet

Pods:

cloud native apps

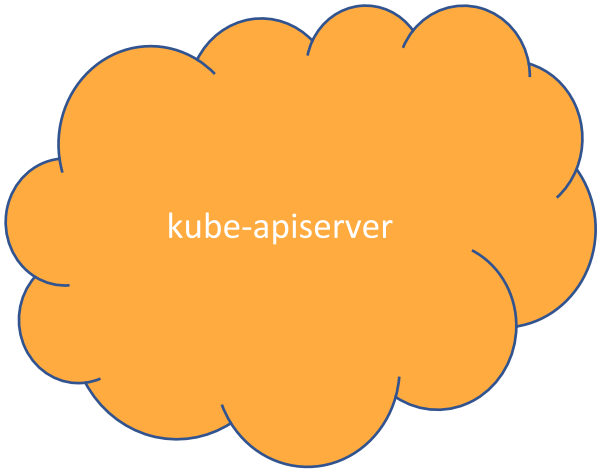
Master:

scheduler

controller manager

ingress controller

apiserver



Restful
http
API

/
/version

/api
/api/v1/**pods**
/api/v1/**pods**/<name>
/api/v1/**pods**/<name>/status

/apis
/apis/batch
/apis/batch/v2alpha1
/apis/batch/v2alpha1/**jobs**
/apis/batch/v2alpha1/**cronjobs**
/apis/batch/v1
/apis/batch/v1/**jobs**

/apis/**<our-group>**/v1/**<our-resource>**

User:

$ kubectl create -f foo.yaml

Node:

proxy

kubelet

Pods:

cloud native apps

Master:

scheduler

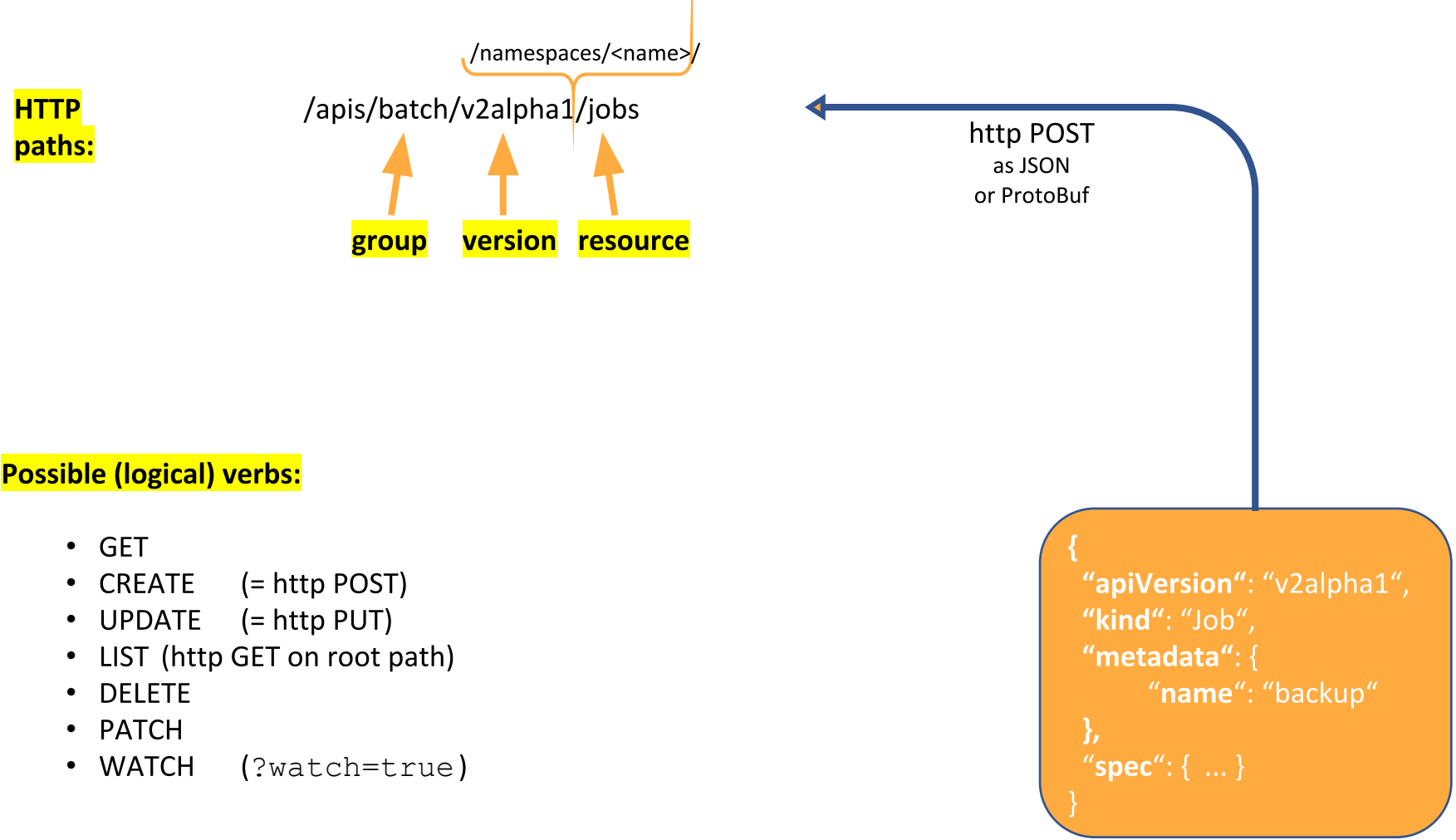
controller manager

ingress controller

apiserver

```
$ kubectl proxy
$ curl http://127.0.0.1:8001/apis
$ kubectl get --raw=/apis

$ kubectl annotate namespace default workshop=gopherconuk
$ kubectl get namespace default  -o json
$ curl http://127.0.0.1:8001/api/v1/namespaces/default
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "default",
    "selfLink": "/api/v1/namespaces/default",
    "uid": "f86e4e1f-94ad-11e8-9d48-0242ac11002b",
    "resourceVersion": "283",
    "creationTimestamp": "2018-07-31T10:39:15Z",
    "annotations": {
      "workshop": "gopherconuk"
    }
  },
  "spec": {
    "finalizers": [
      "kubernetes"
    ]
  },
  "status": {
    "phase": "Active"
  }
}
```

```
$ curl http://127.0.0.1:8001/api/v1/namespaces/default | \
  jq ".metadata.annotations[\"workshop\"] = \"$(date)\"" | \
  curl -H "Content-Type: application/json" -X PUT -d @- http://127.0.0.1:8001/api/v1/namespaces/default
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   430  100   430    0     0  29689      0 --:--:-- --:--:-- --:--:-- 30714
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "default",
    "selfLink": "/api/v1/namespaces/default",
    "uid": "69c4fd62-94a9-11e8-b75a-0242ac11006e",
    "resourceVersion": "2091",
    "creationTimestamp": "2018-07-31T10:06:37Z",
    "annotations": {
      "workshop": "Tue Jul 31 10:32:16 UTC 2018"
    }
  },
  "spec": {
    "finalizers": [
      "kubernetes"
    ]
  },
  "status": {
    "phase": "Active"
  }
}
```



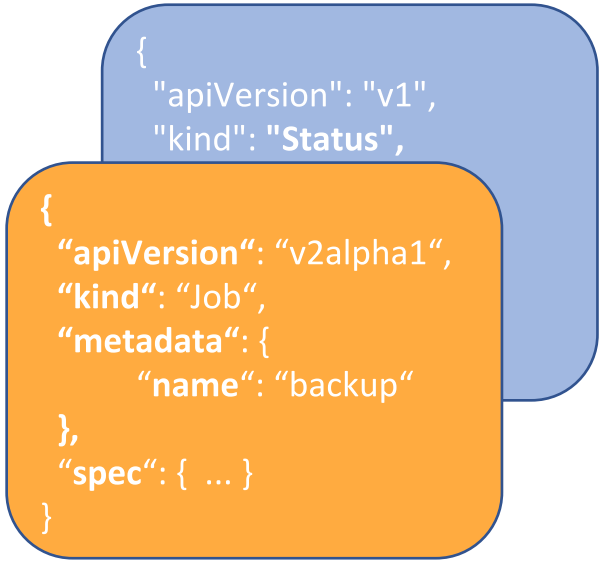
Possible (logical) verbs:

- GET
- CREATE (= http POST)
- UPDATE (= http PUT)
- LIST (http GET on root path)
- DELETE
- PATCH
- WATCH (`?watch=true`)

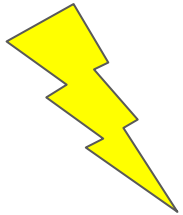
/apis/batch/v2alpha1/jobs

Resource vs. **Kind**

http path vs. **logical object**

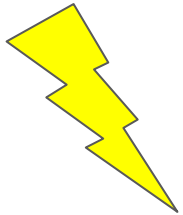


* I omitted the namespace in /apis/batch/v1/jobs/**namespaces/default**/backup



Katacoda

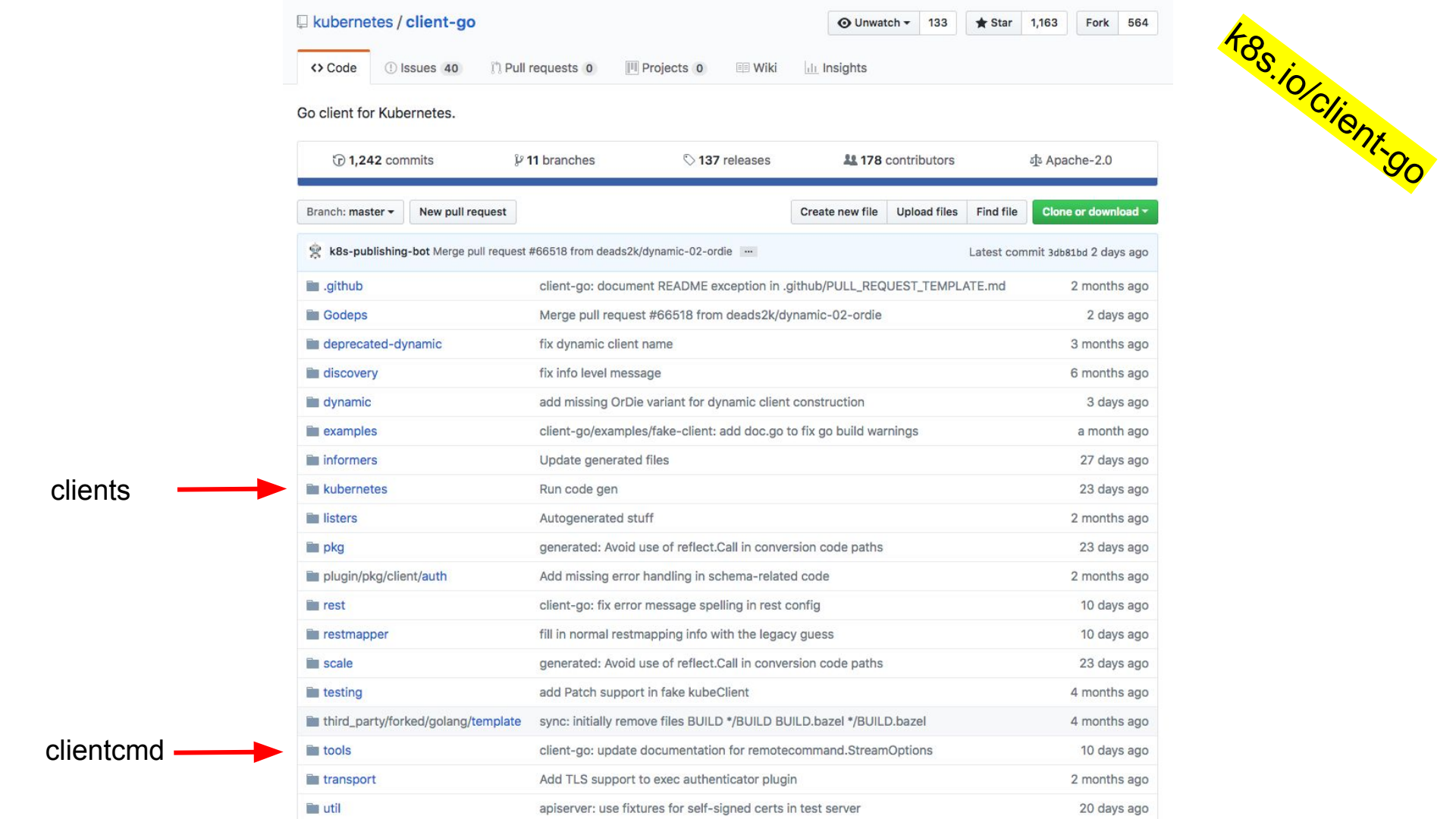
https://301.sh/2018-gopherconuk-katacoda



Katacoda

→ API basics (step 2)

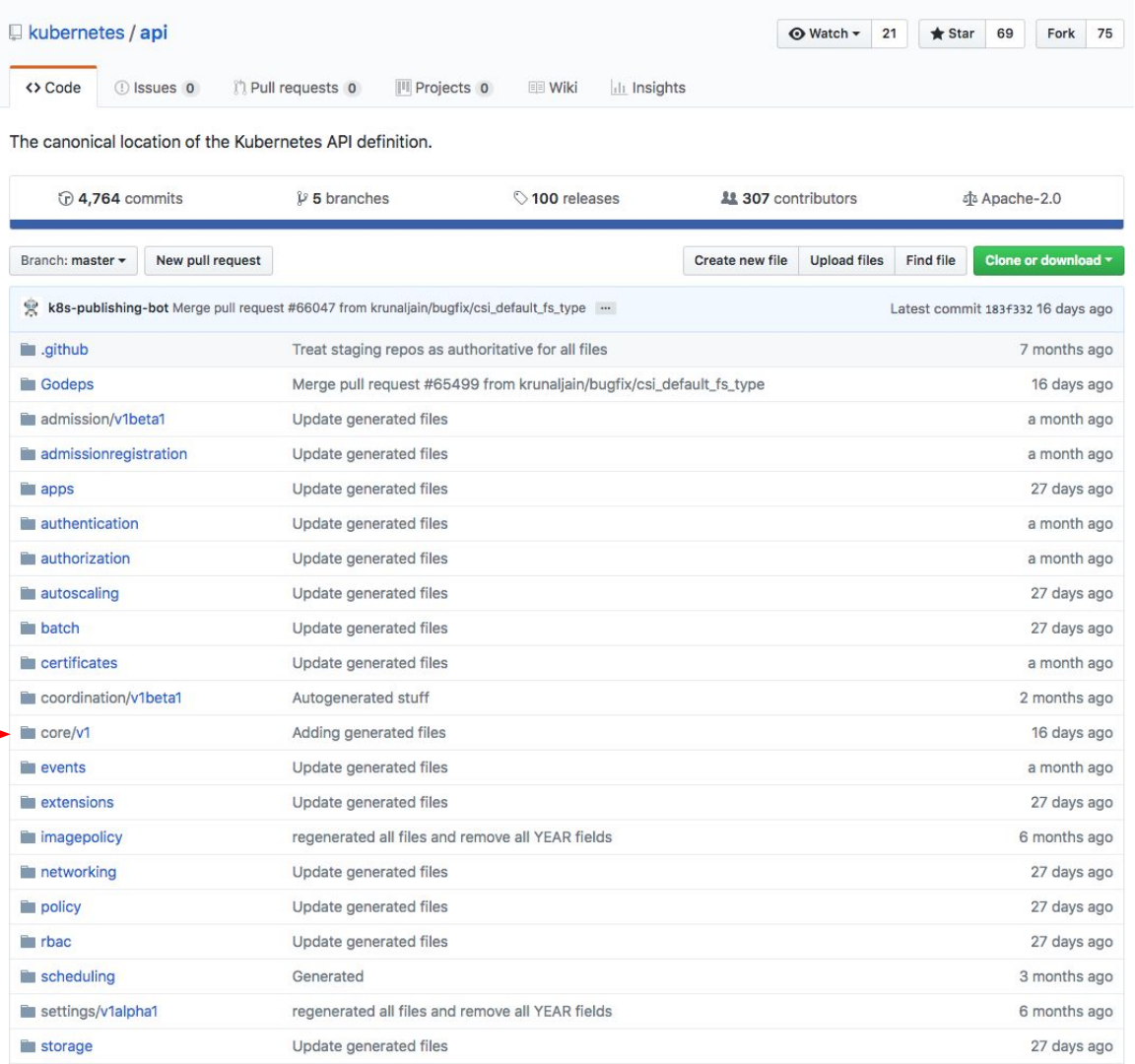
Client-go basics



k8s.io/client-go

clients

clientcmd

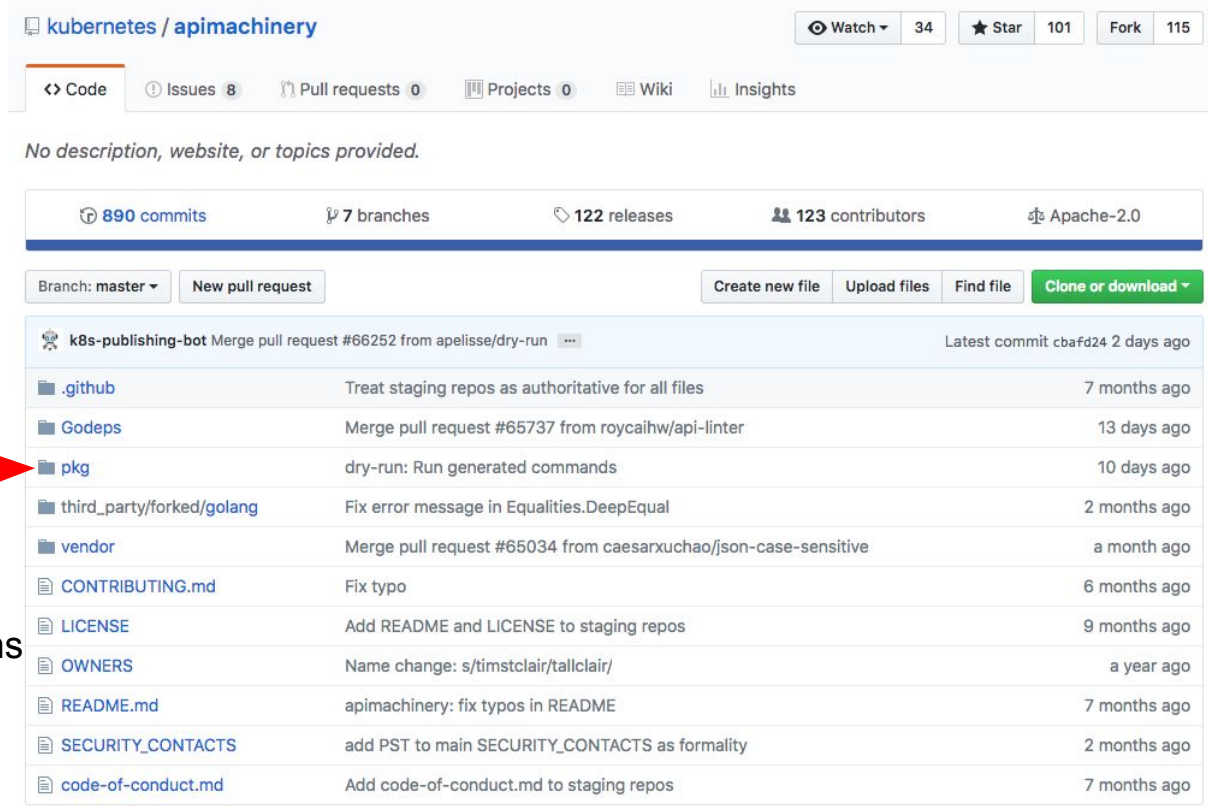


k8s.io/api

core/v1

- Pods
- Services
- ReplicaSet
- ...

k8s.io/apimachinery



pkg/apis/meta/v1

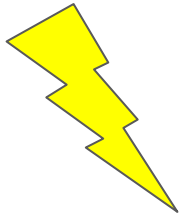
- ObjectMeta
- TypeMeta
- ListOptions
- DeleteOptions
- GetOptions
- Status
- Event
- ...

https://github.com/openshift-talks/k8s-go/blob/master/client-go-basic/main.go

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/tools/clientcmd"
    "k8s.io/client-go/kubernetes"
)

kubeconfig = flag.String("kubeconfig", "~/.kube/config", "path to the kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
clientset, err := kubernetes.NewForConfig(config)

pod, err := clientset.CoreV1().Pods("gophercon").Get("workshop", metav1.GetOptions{})
```

*k8s.io/api/core/v1.Pod

Get, Create, Update, List, Delete, Patch, Watch

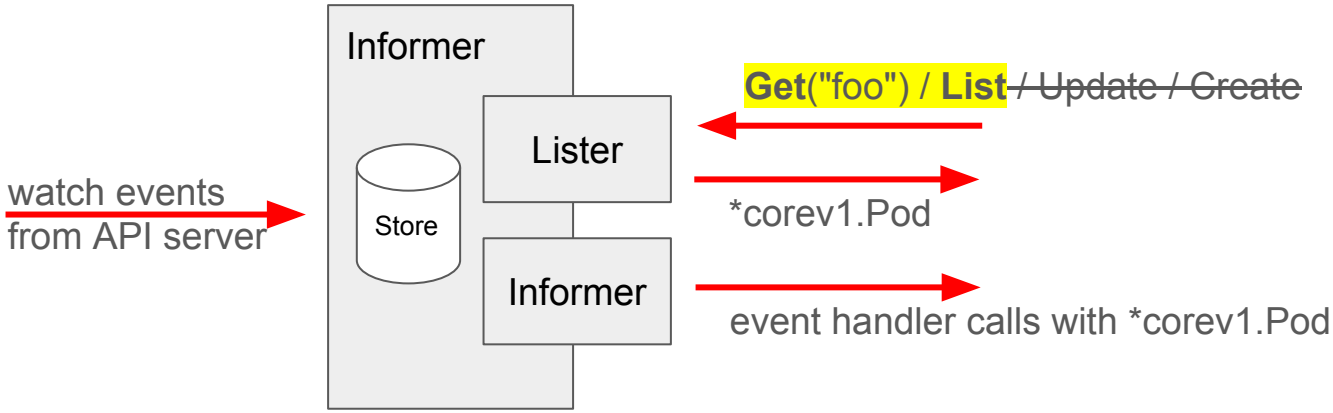


Katacoda

⟶ Client-go basics (step 3)

Informers

k8s.io/client-go/informers



Informers

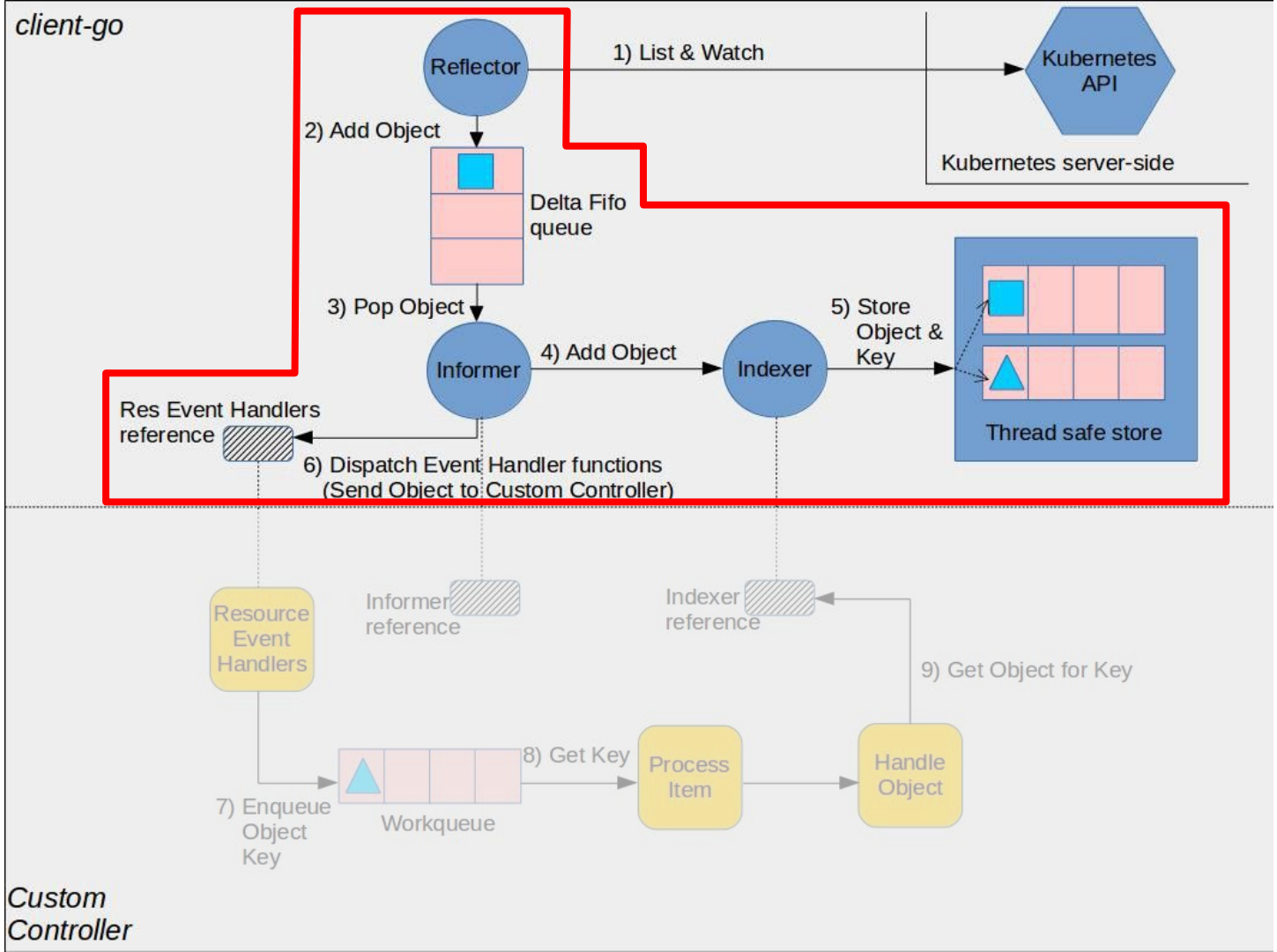


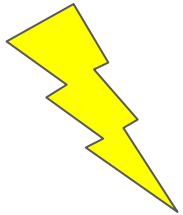
Image by @cloudark / Devdatta Kulkarni
https://medium.com/@cloudark/kubernetes-custom-controllers-b6c7d0668fdf

```
import (
    ...
    "k8s.io/client-go/informers"
)
...
clientset, err := kubernetes.NewForConfig(config)
informerFactory := informers.NewSharedInformerFactory(clientset, time.Second*30)
podInformer := informerFactory.Core().V1().Pods()
podInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
    AddFunc: func(new interface{}) {...},
    UpdateFunc: func(old, new interface{}) {...},
    DeleteFunc: func(obj interface{}) {...},
})
go informerFactory.Start(wait.NeverStop)

pod, err := podInformer.Lister().Pods("gophercon").Get("workshop")
```

resync interval
= how often you re-see your objects, even w/o changes on the server

*k8s.io/api/core/v1.Pod

in-memory cache – updated by List+Watch of the client



Katacoda

⟶ Client-go basics (step 3)

Vendoring client-go – versions and tags

semver

k8s.io/client-go:	v7.0.0	v8.0.0
	=	=
k8s.io/api:	kubernetes-1.10.0	kubernetes-1.11.0
	=	=
k8s.io/apimachinery:	kubernetes-1.10.0	kubernetes-1.11.0

no semver

Compatibility:

	Kubernetes 1.5	Kubernetes 1.6	Kubernetes 1.7	Kubernetes 1.8	Kubernetes 1.9	Kubernetes 1.10	Kubernetes 1.11
client-go 1.5	-	-	-	-	-	-	-
client-go 2.0	✓	+-	+-	+-	+-	+-	+-
client-go 3.0	+-	✓	-	+-	+-	+-	+-
client-go 4.0	+-	+-	✓	+-	+-	+-	+-
client-go 5.0	+-	+-	+-	✓	+-	+-	+-
client-go 6.0	+-	+-	+-	+-	✓	+-	+-
client-go 7.0	+-	+-	+-	+-	+-	✓	+-
client-go 8.0	+-	+-	+-	+-	+-	+-	✓
client-go HEAD	+-	+-	+-	+-	+-	+-	+-

Vendoring client-go – compatibility

- look at the **API groups** and **versions**: are group versions supported by the target cluster?
  - v1alpha1 – unstable:
    - might go away or change any time
    - often disabled by default
  - v1beta1 – towards stable:
    - exists at least one release in parallel to v1
    - no incompatible API change
  - v1 – stable, GA:
    - will stay
    - will be compatible

- look at the **Golang types** and their **fields:**
  
  we have alpha fields in v1, if alpha feature disabled:
  some rejected, some ignored, some dropped

Vendoring client-go

- **source of truth:** Godeps/Godeps.json
- glide reads it
- dep only reads it on init, **not later on ensure**! 

=> specify all dependencies, at least k8s.io/{client-go,api,apimachinery}

dep Gopkg.toml

```
[[constraint]]
  name = "k8s.io/api"
  version = "kubernetes-1.10.0"

[[constraint]]
  name = "k8s.io/apimachinery"
  version = "kubernetes-1.10.0"

[[constraint]]
  name = "k8s.io/client-go"
  version = "7.0.0"

[prune]
  go-tests = true
  unused-packages = true
```

+

```
# the following overrides are necessary to enforce
# the given version, even though our
# code does not import the packages directly.
[[override]]
  name = "k8s.io/api"
  version = "kubernetes-1.10.0"

[[override]]
  name = "k8s.io/apimachinery"
  version = "kubernetes-1.10.0"

[[override]]
  name = "k8s.io/client-go"
  version = "7.0.0"
```

Kubernetes objects in Go

Golang Types

k8s.io/api/core/v1/types.go

```
type Pod struct {
     metav1.TypeMeta   `json:",inline"`
     metav1.ObjectMeta `json:"metadata,omitempty"`

     Spec   SessionSpec   `json:"spec"`
     Status SessionStatus `json:"status"`
}

type PodSpec struct {
     Volumes []Volume `json:"volumes,omitempty"`
     Containers []Container `json:"containers"`
     NodeName string `json:"nodeName,omitempty"`
}

type PodStatus struct {
     Phase PodPhase `json:"phase,omitempty"`
     Conditions []PodCondition `json:"conditions,omitempty"`
     ContainerStatuses []ContainerStatus `json:"containerStatuses,omitempty"`
}
```

user's desire

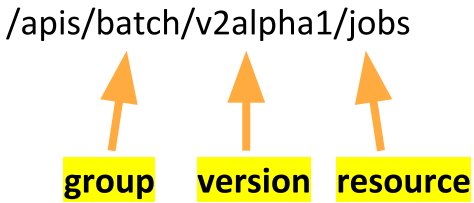
how the controller saw it last time

```
type TypeMeta struct {
    Kind string `json:"kind,omitempty" protobuf:"bytes,1,opt,name=kind"`
    APIVersion string `json:"apiVersion,omitempty" protobuf:"bytes,2,opt,name=apiVersion"`
}

type ObjectMeta struct {
    Name                       string           `json:"name,omitempty" protobuf:"bytes,1,opt,na
    GenerateName               string           `json:"generateName,omitempty" protobuf:"bytes,
    Namespace                  string           `json:"namespace,omitempty" protobuf:"bytes,3,o
    SelfLink                   string           `json:"selfLink,omitempty" protobuf:"bytes,4,op
    UID                        types.UID        `json:"uid,omitempty" protobuf:"bytes,5,opt,nam
    ResourceVersion            string           `json:"resourceVersion,omitempty" protobuf:"byt
    Generation                 int64            `json:"generation,omitempty" protobuf:"varint,7
    CreationTimestamp          Time             `json:"creationTimestamp,omitempty" protobuf:"b
    DeletionTimestamp          *Time            `json:"deletionTimestamp,omitempty" protobuf:"b
    DeletionGracePeriodSeconds *int64           `json:"deletionGracePeriodSeconds,omitempty" pr
    Labels                     map[string]string `json:"labels,omitempty" protobuf:"bytes,11,rep
    Annotations                map[string]string `json:"annotations,omitempty" protobuf:"bytes,1
    OwnerReferences            []OwnerReference  `json:"ownerReferences,omitempty" patchStrategy
    Initializers               *Initializers    `json:"initializers,omitempty" protobuf:"bytes,
    Finalizers                 []string         `json:"finalizers,omitempty" patchStrategy:"mer
    ClusterName                string           `json:"clusterName,omitempty" protobuf:"bytes,1
}
```

GVK and GVR

runtime.**GroupVersionKind**{**Group**: "batch", **Version**: "v2alpha1", **Kind:** "Job"}

runtime.**GroupVersionResource**{**Group**: "batch", **Version**: "v2alpha1", **Resource**: "jobs"}



Sometimes also: GroupVersion, GroupKind, GroupResource, ...

Golang types boilerplate

k8s.io/api/core/v1/register.go

```
const GroupName = "" // the legacy core group. Normally this would be e.g. apps.k8s.io
var SchemeGroupVersion = schema.GroupVersion{Group: GroupName, Version: "v1"}

func Kind(kind string) schema.GroupKind {
   return SchemeGroupVersion.WithKind(kind).GroupKind()
}

func Resource(resource string) schema.GroupResource {
   return SchemeGroupVersion.WithResource(resource).GroupResource()
}

var SchemeBuilder = runtime.NewSchemeBuilder(addKnownTypes)
var AddToScheme   = SchemeBuilder.AddToScheme

// Adds the list of known types to Scheme.
func addKnownTypes(scheme *runtime.Scheme) error {
   scheme.AddKnownTypes(SchemeGroupVersion,
      &Pod{},
      &PodList{},
   )
   metav1.AddToGroupVersion(scheme, SchemeGroupVersion)
   return nil
}
```

Install some generic types like Status, GetOptions/ListOptions/DeleteOptions

Scheme – k8s.io/apimachinery/pkg/runtime.Scheme

k8s.io/client-go/kubernetes/scheme for a scheme with all Kube types

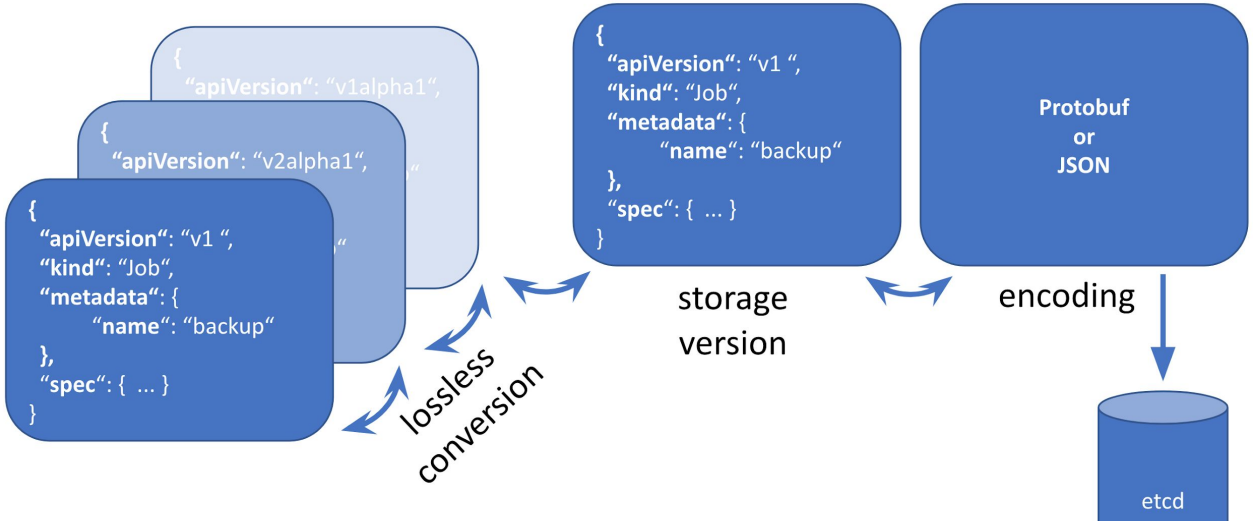
```
type Scheme struct {
    // versionMap allows one to figure out the go type of an object with
    // the given version and name.
    gvkToType map[schema.GroupVersionKind]reflect.Type

    // typeToGroupVersion allows one to find metadata for a given go object.
    // The reflect.Type we index by should *not* be a pointer.
    typeToGVK map[reflect.Type][]schema.GroupVersionKind

    // defaulterFuncs is an array of interfaces to be called with an object to provide defaulting
    // the provided object must be a pointer.
    defaulterFuncs map[reflect.Type]func(interface{})

    // converter stores all registered conversion functions. It also has
    // default coverting behavior.
    converter *conversion.Converter
}

func (s *Scheme) AddKnownTypes(gv schema.GroupVersion, types ...Object) {
```

Conversion – `scheme.Convert`

Only relevant in the API server implementation

```
func (s *Scheme) Convert(in, out interface{}, context interface{}) error
```

- used by API servers internally to convert from e.g. **v1** objects to **v1beta1**
- … usually by converting from **v1** to **"_internal"** to **v1beta**



Defaulting – `scheme.Default`

Only relevant in the API server implementation

```
func (s *Scheme) Default(src Object)
```

- done by the API server for all incoming objects and on read from etcd

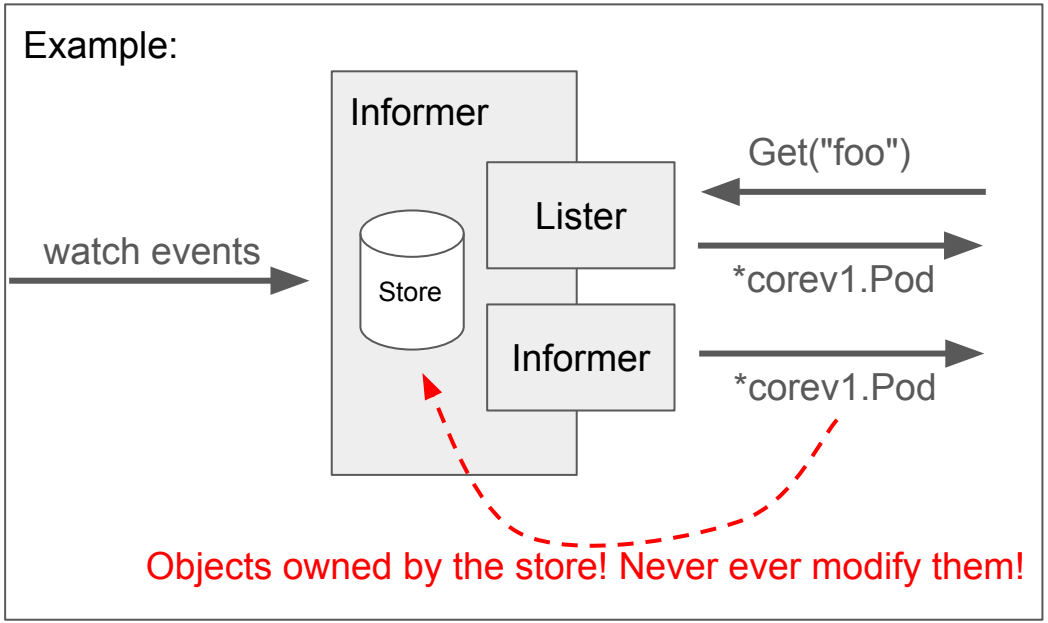
sets fields that are unset

- example: **`PodSpec.Containers[*].RestartPolicy`**

  If unset, defaults to **RestartPolicyAlways**.

DeepCopy- `obj.DeepCopy()`

- generated for all types




```go
func (in *Pod) DeepCopyInto(out *Pod) {
    *out = *in
    out.TypeMeta = in.TypeMeta
    in.ObjectMeta.DeepCopyInto(&out.ObjectMeta)
    in.Spec.DeepCopyInto(&out.Spec)
    in.Status.DeepCopyInto(&out.Status)
    return
}

func (in *Pod) DeepCopy() *Pod {
    if in == nil {
        return nil
    }
    out := new(Pod)
    in.DeepCopyInto(out)
    return out
}

func (in *Pod) DeepCopyObject() runtime.Object {
    if c := in.DeepCopy(); c != nil {
        return c
    }
    return nil
}
```

RESTMapper

≈ GroupVersionResource + cluster/namespace scope

```
type RESTMapper interface {
    RESTMapping(gk schema.GroupKind, versions ...string) (*RESTMapping, error)

    KindFor(resource schema.GroupVersionResource) (schema.GroupVersionKind, error)
    ResourceFor(input schema.GroupVersionResource) (schema.GroupVersionResource, error)
}
```

Often used: DeferredDiscoveryRESTMapper with discovery

```
cachedClient := cacheddiscovery.NewMemCacheClient(clientset.Discovery())
restMapper := restmapper.NewDeferredDiscoveryRESTMapper(cachedClient)
```

Custom Resources (CRs)
&
Custom Resource Definitions (CRDs)

apiextensions/v1beta1

Custom Resource

example-healthcheckpolicy.yaml

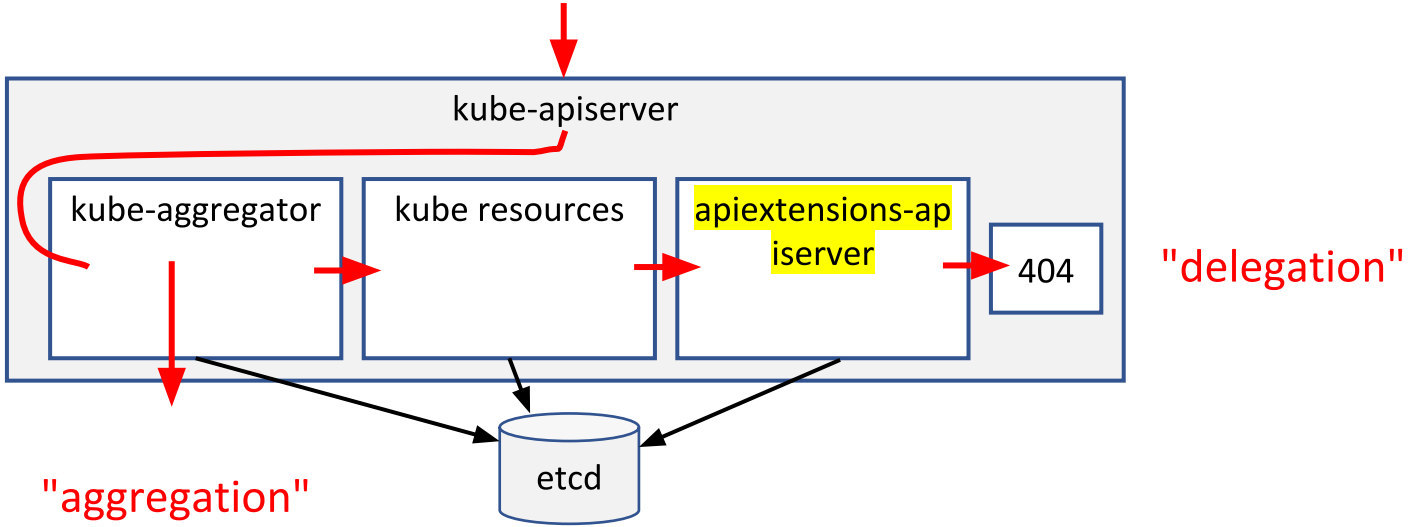
```
apiVersion: policy.k8s-go.openshift.org/v1alpha1
kind: HealthCheckPolicy
metadata:
  name: example
spec:
  podLabel: health-check-missing
status:
  statistics:
    pods:
      failed: 5
      passed: 42
  lastChecked: "So 29 Jul 2018 09:13:07 CEST"
```

We want to store these

apiextensions-apiserver inside kube-apiserver

https://github.com/kubernetes/apiextensions-apiserver

In every Kubernetes 1.7+ cluster.



Custom Resource

example-healthcheckpolicy.yaml

```
apiVersion: policy.k8s-go.openshift.org/v1alpha1
kind: HealthCheckPolicy
metadata:
  name: example
spec:
  podLabel: health-check-missing
status:
  statistics:
    pods:
      failed: 5
      passed: 42
  lastChecked: "So 29 Jul 2018 09:13:07 CEST"
```

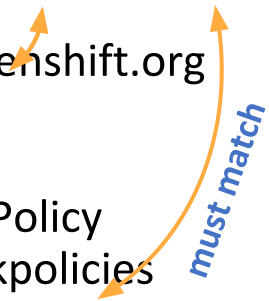
We want to store these

Custom Resource Definition (CRD)

apiextensions/v1beta1

Defines how CRs are stored

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
    name: healthcheckpolicies.policy.k8s-go.openshift.org
spec:
    group: policy.k8s-go.openshift.org
    version: v1alpha1
    names:
        kind: HealthCheckPolicy
        plural: healthcheckpolicies
        scope: Namespaced
```



Custom Resource Definition (CRD)

apiextensions/v1beta1

$ kubectl **create** –f healthcheckpolicy-crd.yaml

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
    name: healthcheckpolicies.policy.k8s-go.openshift.org
spec:
    group: policy.k8s-go.openshift.org
    version: v1alpha1
    names:
        kind: HealthCheckPolicy
        plural: healthcheckpolicies
        scope: Namespaced
```

must match

a moment

```
...
status:
acceptedNames:
        kind: HealthCheckPolicy
        listKind: HealthCheckPolicyList
        plural: healthcheckpolicies
        singular: healthcheckpolicy
conditions:
    - type: NamesAccepted
      message: no conflicts found
      reason: NoConflicts
      status: "True“
    - type: Established
      message: the initial names have
          been accepted
      reason: InitialNamesAccepted
      status: "True“
```

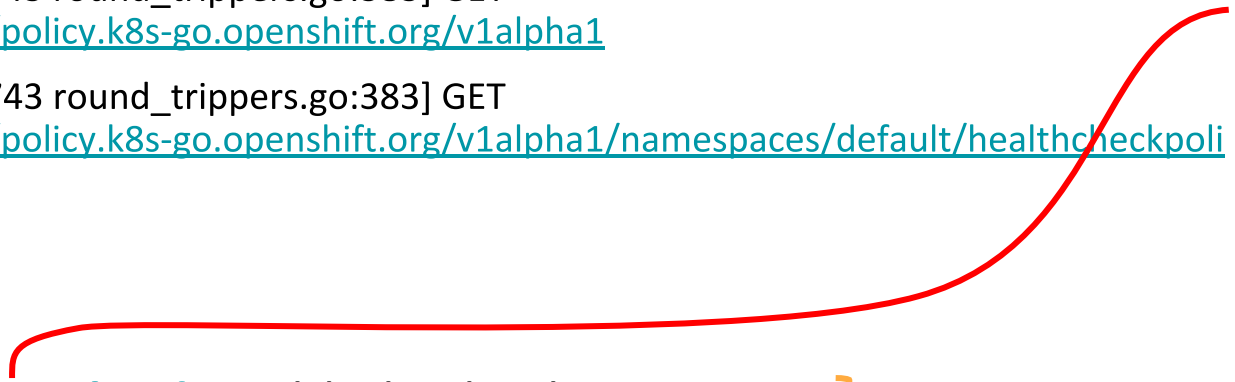
$ kubectl get healthcheckpolicies –v=7

- I0429 21:17:53.042783 66743 round_trippers.go:383] GET https://localhost:6443/apis
- I0429 21:17:53.135811 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1
- I0429 21:17:53.138353 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1/namespaces/default/healthcheckpolicies

discovery

LIST

No resources found.



healthcheckpolicies → **kind** HealthCheckPolicy
resource healthcheckpolicies

API group
policy.k8s-go.openshift.org/v1

We call this "REST mapping"

$ http localhost:8080/apis/

```
{
        "groups": [{
                "name": "policy.k8s-go.openshift.org",
                "preferredVersion": {"groupVersion": "policy.k8s-go.openshift.org/v1", "version": "v1alpha1“},
                "versions": [{"groupVersion": "policy.k8s-go.openshift.org/v1alpha1",  "version": "v1alpha1"}]
        }, ...]
}
```

$ http localhost:8080/apis/policy.k8s-go.openshift.org/v1alpha1

```
{
        "apiVersion": "v1",
        "groupVersion": "policy.k8s-go.openshift.org/v1alpha1",
        "kind": "APIResourceList",
        "resources": [{
                "kind": "HealthCheckPolicy",
                "name": "healthcheckpolicies",
                "namespaced": true,
                "verbs": ["create", "delete", "deletecollection",
                        "get", "list", "patch", "update", "watch“
                ]
        }, ...]
}
```

resource name ⇒ /apis/policy.k8s-go.openshift.org/v1alpha1/healthcheckpolicies

/namespaces/<name>/

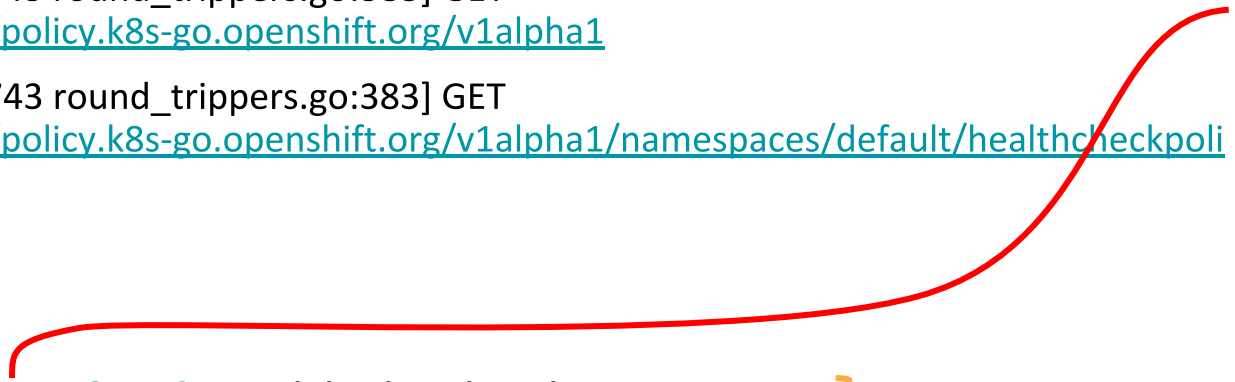
$ kubectl get healthcheckpolicies –v=7

- I0429 21:17:53.042783 66743 round_trippers.go:383] GET https://localhost:6443/apis
- I0429 21:17:53.135811 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1
- I0429 21:17:53.138353 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1/namespaces/default/healthcheckpolicies

discovery

LIST

No resources found.



healthcheckpolicies → **kind** HealthCheckPolicy
resource healthcheckpolicies

API group
policy.k8s-go.openshift.org/v1

We call this "REST mapping"

$ kubectl get **healthcheckpolicies** –v=7

- I0429 21:17:53.042783 66743 round_trippers.go:383] GET https://localhost:6443/apis
- I0429 21:17:53.135811 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1
- I0429 21:17:53.138353 66743 round_trippers.go:383] GET https://localhost:6443/apis/policy.k8s-go.openshift.org/v1alpha1/namespaces/default/healthcheckpolicies

discovery

LIST

No resources found.

hcp → **kind** HealthCheckPolicy
resource healthcheckpolicies

API group
policy.k8s-go.openshift.org/v1

note: a "shortName"

We call this "REST mapping"

1

Create the CRD

$ kubectl create –f healthcheckpolicies-crd.yaml

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
        name: healthcheckpolicies.policy.k8s-go...
spec:
        group: policy.k8s-go.openshift.org
        version: v1alpha1
        names:
                kind: HealthCheckPolicy
                plural: healthcheckpolicies
                scope: Namespaced
```

2

Create CustomResources

$ kubectl create –f example-healthcheckpolicy.yaml

```
apiVersion:
policy.k8s-go.openshift.org/v1alpha1
kind: HealthCheckPolicy
metadata:
  name: example
spec:
  podLabel: health-check-missing
status:
  statistics:
    pods:
      failed: 5
      passed: 42
  lastChecked: "So 29 Jul 2018 09:13:07 CEST"
```

1

Create the CRD

$ kubectl create –f healthcheckpolicies-crd.yaml

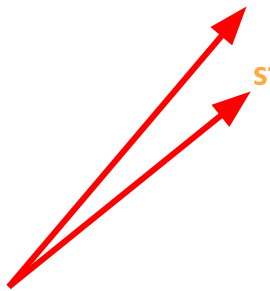
```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
        name: healthcheckpolicies.policy.k8s-go...
spec:
        group: policy.k8s-go.openshift.org
        version: v1alpha1
        names:
                kind: HealthCheckPolicy
                plural: healthcheckpolicies
                scope: Namespaced
```

2

Create CustomResources

$ kubectl create –f example-healthcheckpolicy.yaml

```
apiVersion:
policy.k8s-go.openshift.org/v1alpha1
kind: HealthCheckPolicy
metadata:
  name: example
spec:
  podLabel: health-check-missing
status:
  statistics:
    pods:
      failed: 5
      passed: 42
  lastChecked: "So 29 Jul 2018 09:13:07 CEST"
```



Follow **spec+status pattern**.

Watch, i.e. event stream

$ kubectl get healthcheckpolicies **–w** --no-headers
example <none> {"apiVersion":"policy.k8s-go.openshift.org/v1","kind":"HealthCheckPolicies",...
example2 <none> {"apiVersion":"policy.k8s-go.openshift.org/v1","kind":" HealthCheckPolicies ",...

$ curl -f
'http://127.0.0.1:8080/apis/policy.k8s-go.openshift.org/v1/namespaces/default/healthcheckpolicies?**watch=true**&**resourceVersion=434**‘

{"type":"DELETED","object":{"apiVersion":"policy.k8s-go.openshift.org/v1","kind":"HealthCheckPolicies","metadata":{"name":"example","namespace":"default","selfLink":"/apis/policy.k8s-go.openshift.org/v1/namespaces/default/healthcheckpolicies/example","uid":"8f5312c0-29c8-11e7-88f9-4c3275978b79","resourceVersion":"435","creationTimestamp":"2017-04-25T15:05:03Z"},"spec":{...}}}

{"type":"ADDED","object":{"apiVersion":"policy.k8s-go.openshift.org/v1","kind":"HealthCheckPolicies","metadata":{"name":"example2","namespace":"default","selfLink":"/apis/policy.k8s-go.openshift.org/v1/namespaces/default/healthcheckpolicies/example2","uid":"b4318cb5-29c8-11e7-88f9-4c3275978b79","resourceVersion":"436","creationTimestamp":"2017-04-25T15:06:05Z"},"spec":{...}}}

Validation

- The standard: OpenAPI v3 schema
- based on JSON Schema

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
        name: healthcheckpolicies.policy.k8s-go...
spec:
        group: policy.k8s-go.openshift.org
        version: v1alpha1
        names: …
        validation:
           openAPIV3Schema: <see next slide>
```

Custom Resource

```yaml
spec:
  action: label
  podLabel: health-check-missing
status:
  statistics:
    pods:
      failed: 5
      passed: 42
  lastChecked: "So 29 Jul 2018 09:13:07..."
```

```yaml
properties:
  spec:
    properties:
      action:
        anyOf: [{"pattern": "^label|kill|annotate$"}, ...]
        default: "label"
      podLabel: {"type": "string", "pattern": "^[0-9]+-...$"}
    required: ["action"]
  status:
    properties:
      statistics:
        properties:
          failed: {"type": "integer", "minimum": "0"}
          passed: {"type": "integer", "minimum": "0"}
    required: []
```

a quantor (anyOf, oneOf, allOf exist)

note: enum is forbidden (why?)

probably in 1.12+

regular expression

OpenAPI v3 Schema

Helpful tools:

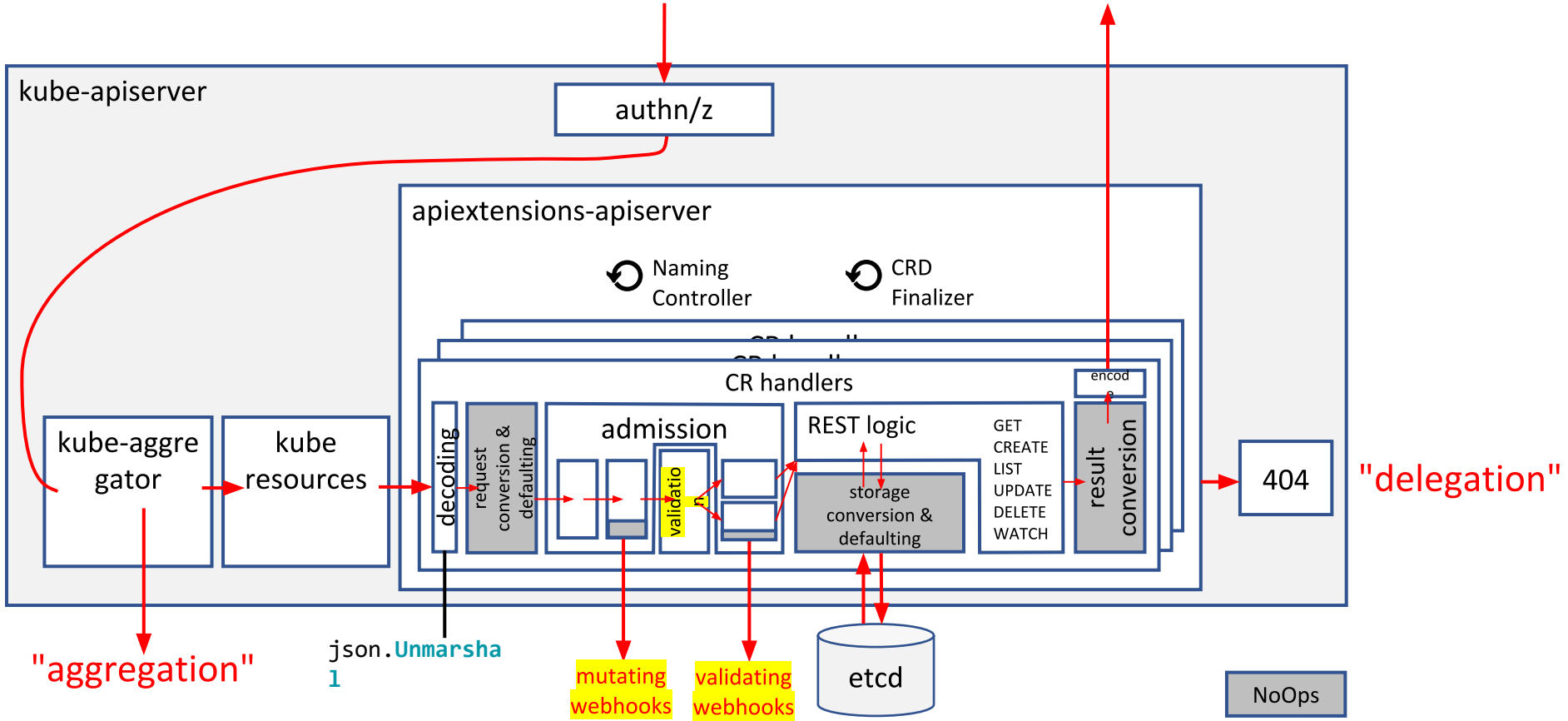
kubernetes/kube-openapi#37

tamalsaha/kube-openapi-generator

Some other tool from prometheus-operator?

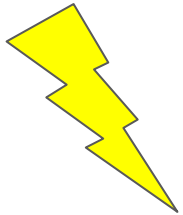
Rancher has another one, speak to @lemonjet

Zoom into apiextensions-apiserver



"CRDs are limited"

- no version conversion (only one version possible per CRD)
  - in 1.11 multiple versions, but no conversion
  - in 1.12+ conversions (via webhook) come, but slowly (i.e. first alpha)
- no defaulting (maybe in 1.12 as alpha)
- no validation (beta in 1.9, Google Summer of Code project)
- no subresources (scale, status) (beta in 1.11)
- no admission (since 1.7: admission webhooks + initializers)
- alpha CRDs are beta
- no custom printing in kubectl (in 1.11: custom printer columns)



Katacoda

⟶ CRDs for users (step 4)

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}

type HealthCheckPolicySpec struct {
}

type HealthCheckPolicyStatus struct {
      PodsFailed int64 `json:"podsFailed,omitempty"`
}
```

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}

type HealthCheckPolicySpec struct {
}

type HealthCheckPolicyStatus struct {
      PodsFailed int64 `json:"podsFailed,omitempty"`
}
```

doc.go:

```
// +k8s:deepcopy-gen=package
package v1
```

More info about code generation in my OpenShift blog post [Code Generation for CustomResources]

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}

type HealthCheckPolicySpec struct {
}

type HealthCheckPolicyStatus struct {
      PodsFailed int64 `json:"podsFailed,omitempty"`
}
```

doc.go:

```
// +k8s:deepcopy-gen=package
package v1
```

More info about code generation in my OpenShift blog post [Code Generation for CustomResources]

Interface runtime.Object

```
type Object interface {
  GetObjectKind() schema.ObjectKind
  DeepCopyObject() Object
}
```

= everything that has TypeMeta

```
func (p *Pod) DeepCopyObject() runtime.Object {
 if c := in.DeepCopy(); c != nil {
      return c
 }
 return nil
}
```

Go riddle: why?

generated through: `// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object`

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}

type HealthCheckPolicySpec struct {
}

type HealthCheckPolicyStatus struct {
      PodsFailed int64 `json:"podsFailed,omitempty"`
}
```

doc.go:

```
// +k8s:deepcopy-gen=package
package v1
```

More info about code generation in my OpenShift blog post [Code Generation for CustomResources]

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
// +genclient
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}

type HealthCheckPolicySpec struct {
}

type HealthCheckPolicyStatus struct {
      PodsFailed int64 `json:"podsFailed,omitempty"`
}
```

doc.go:

```
// +k8s:deepcopy-gen=package
package v1
```

More info about code generation in my OpenShift blog post [Code Generation for CustomResources]

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/types.go

```
// +genclient
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type HealthCheckPolicy struct {
      metav1.TypeMeta   `json:",inline"`
      metav1.ObjectMeta `json:"metadata,omitempty"`

      Spec   HealthCheckPolicySpec   `json:"spec"`
      Status HealthCheckPolicyStatus `json:"status"`
}
```

doc.go:

```
// +k8s:deepcopy-gen=package
package v1
```

```
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
type HealthCheckPolicyList struct {
      metav1.TypeMeta `json:",inline"`
      metav1.ListMeta `json:"metadata"`

      Items []HealthCheckPolicy `json:"items"`
}

type HealthCheckPolicySpec struct {
}
```

More info about code generation in my OpenShift blog post [Code Generation for CustomResources]

Towards a controller: Golang types

pkg/apis/policy/v1alpha1/register.go

```
const GroupName = "policy.k8s-go.openshift.org"

var SchemeGroupVersion = schema.GroupVersion{Group: GroupName, Version: "v1alpha1"}

func Kind(kind string) schema.GroupKind {
   return SchemeGroupVersion.WithKind(kind).GroupKind()
}

func Resource(resource string) schema.GroupResource {
   return SchemeGroupVersion.WithResource(resource).GroupResource()
}

var SchemeBuilder = runtime.NewSchemeBuilder(addKnownTypes)
var AddToScheme   = SchemeBuilder.AddToScheme

// Adds the list of known types to Scheme.
func addKnownTypes(scheme *runtime.Scheme) error {
   scheme.AddKnownTypes(SchemeGroupVersion,
      &HealthCheckPolicy{},
      &HealthCheckPolicies{},
   )
   metav1.AddToGroupVersion(scheme, SchemeGroupVersion)
   return nil
}
```

hint: copy from some other API group

Code Generation

all = deepcopy,defaulter,client,lister,informer

```
$ vendor/k8s.io/code-generator/generate-groups.sh all \
  github.com/openshift/k8s-go/crd-go/pkg/generated \
  github.com/openshift/k8s-go/crd-go/pkg/pkg/apis \
  "policy:v1alpha1,v2,v3 someothergroup.io:v1" \
  --output-base "$GOPATH/src" \
  --go-header-file hack/boilerplate.go.txt
```

Usually put into **hack/update-codegen.sh** (compare k8s.io/sample-controller).

doc.go for crazy group and package names:

```
// +groupName=example.test.crd.code-generator.k8s.io
// +groupGoName=SecondExample
```

Alternatives later: kubebuilder, operator-sdk

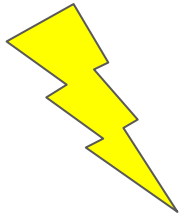
Native CRD Client

```
import (
     metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
     "k8s.io/client-go/tools/clientcmd"
     policyclient "github.com/openshift/k8s-go/crd-go/pkg/generated/clientset/versioned"
)
```

similarly /informers and /listers

```
kubeconfig = flag.String("kubeconfig", "~/.kube/config", "path to the kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
clientset, err := kubecon.NewForConfig(config)

policy, err := clientset.PolicyV1alpha1().HealthCheckPolicies("default").Get("example", {})
```

*github.com/openshift/k8s-go/crd-go/pkg/apis/policy/v1alpha1.HealthCheckPolicy

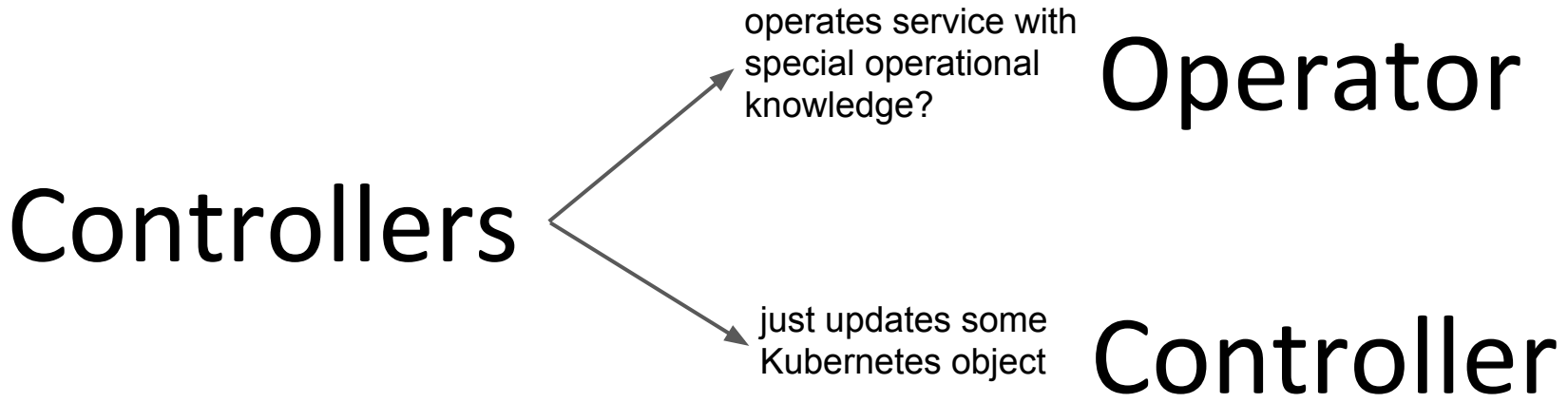


Katacoda

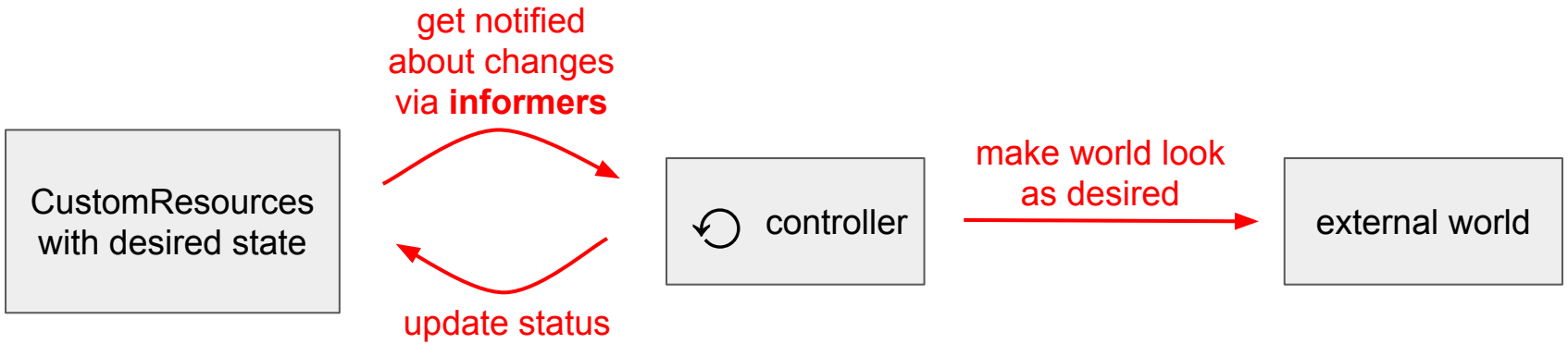
⟶ CRDs for developers (step 5)

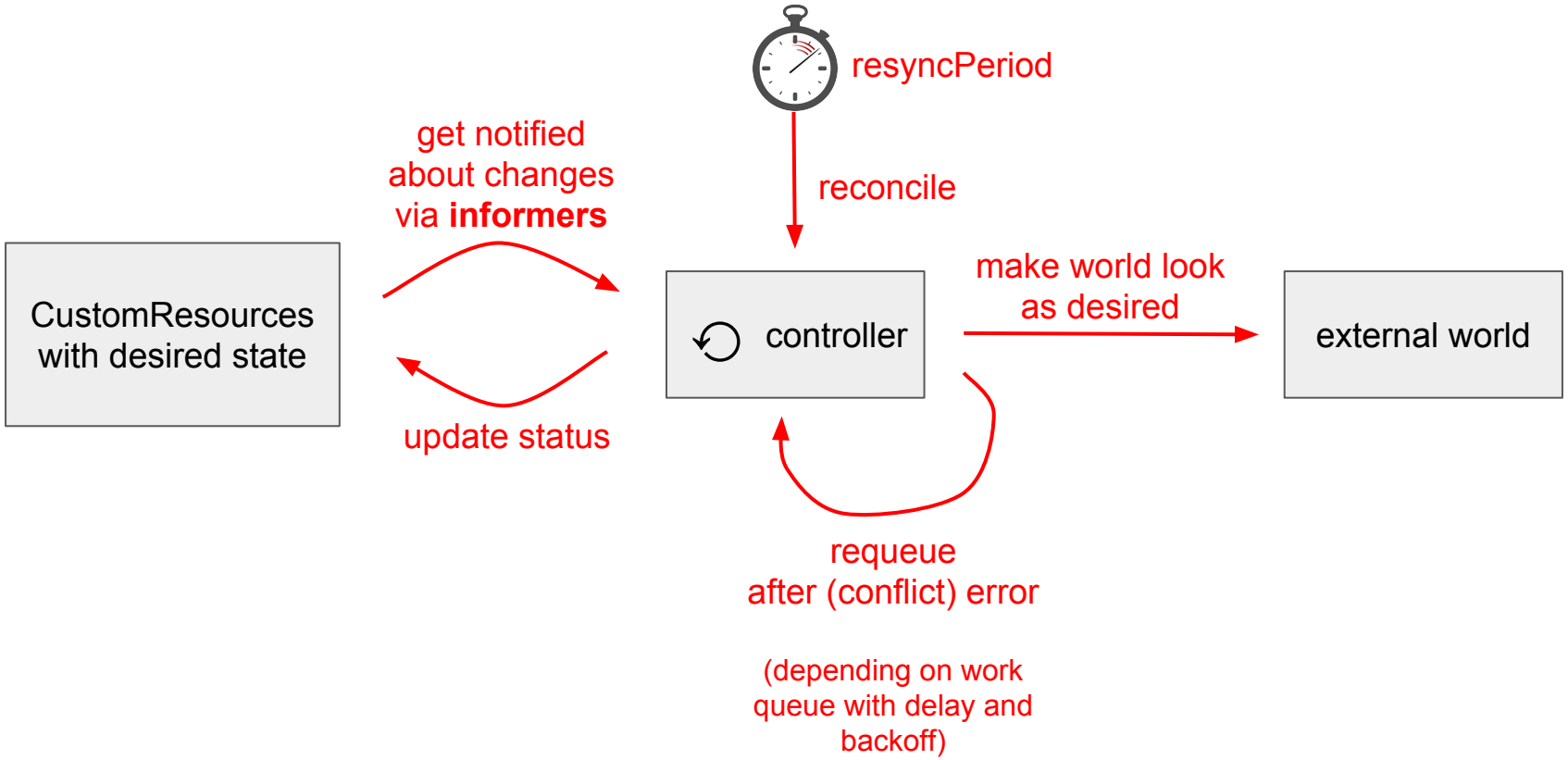
Outlook – Custom Resources

- Kubernetes 1.11+
  - α: **Multiple versions without conversion** – design proposal
  - α: **Server Side Printing Columns** – "kubectl get" customization – #60991
  - β: **Subresources** – α since 1.10 – #62786
  - OpenAPI `additionalProperties` allowed now (mutually exclusive with `properties`)
- Kubernetes 1.12
  - α: **Conversion Webhooks**
  - α: **Pruning** – in validation spec unspecified fields are removed – blocker for GA
  - α: **Defaulting** – defaults from OpenAPI validation schema are applied


Controllers
operates service with special operational knowledge?
Operator
just updates some Kubernetes object
Controller

1. **read** object (preferably event driven with **watches**)
2. **change** object, update the world
3. **update** object on apiserver
4. **repeat**


get notified
about changes
via **informers**
CustomResources
with desired state
controller
make world look
as desired
external world
update status


resyncPeriod
reconcile
get notified
about changes
via **informers**
CustomResources
with desired state
update status
controller
make world look
as desired
external world
requeue
after (conflict) error
(depending on work
queue with delay and
backoff)

Inner workings of a Controller

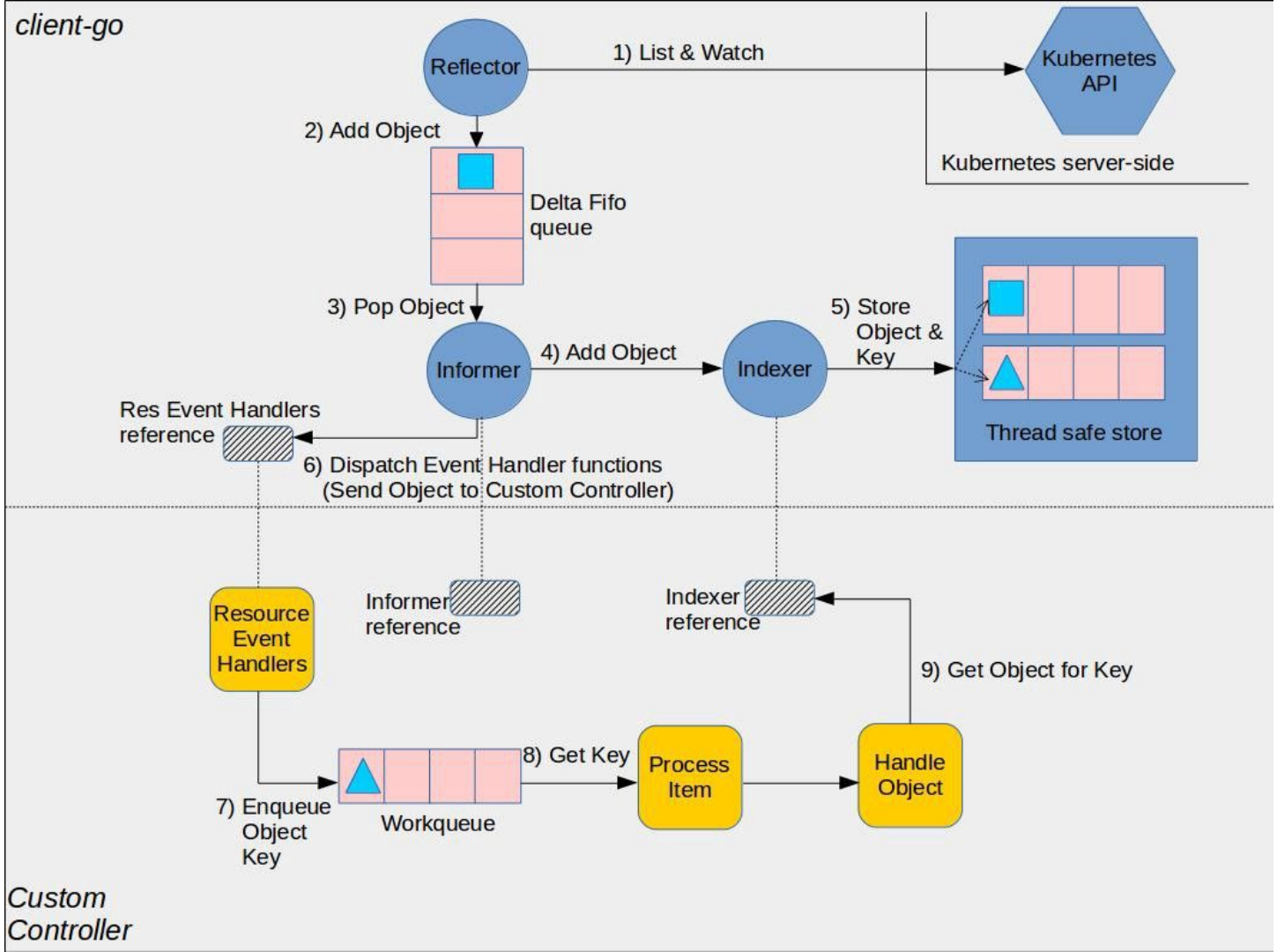


Image by @cloudark / Devdatta Kulkarni
https://medium.com/@cloudark/kubernetes-custom-controllers-b6c7d0668fdf

Inner workings of a Controller

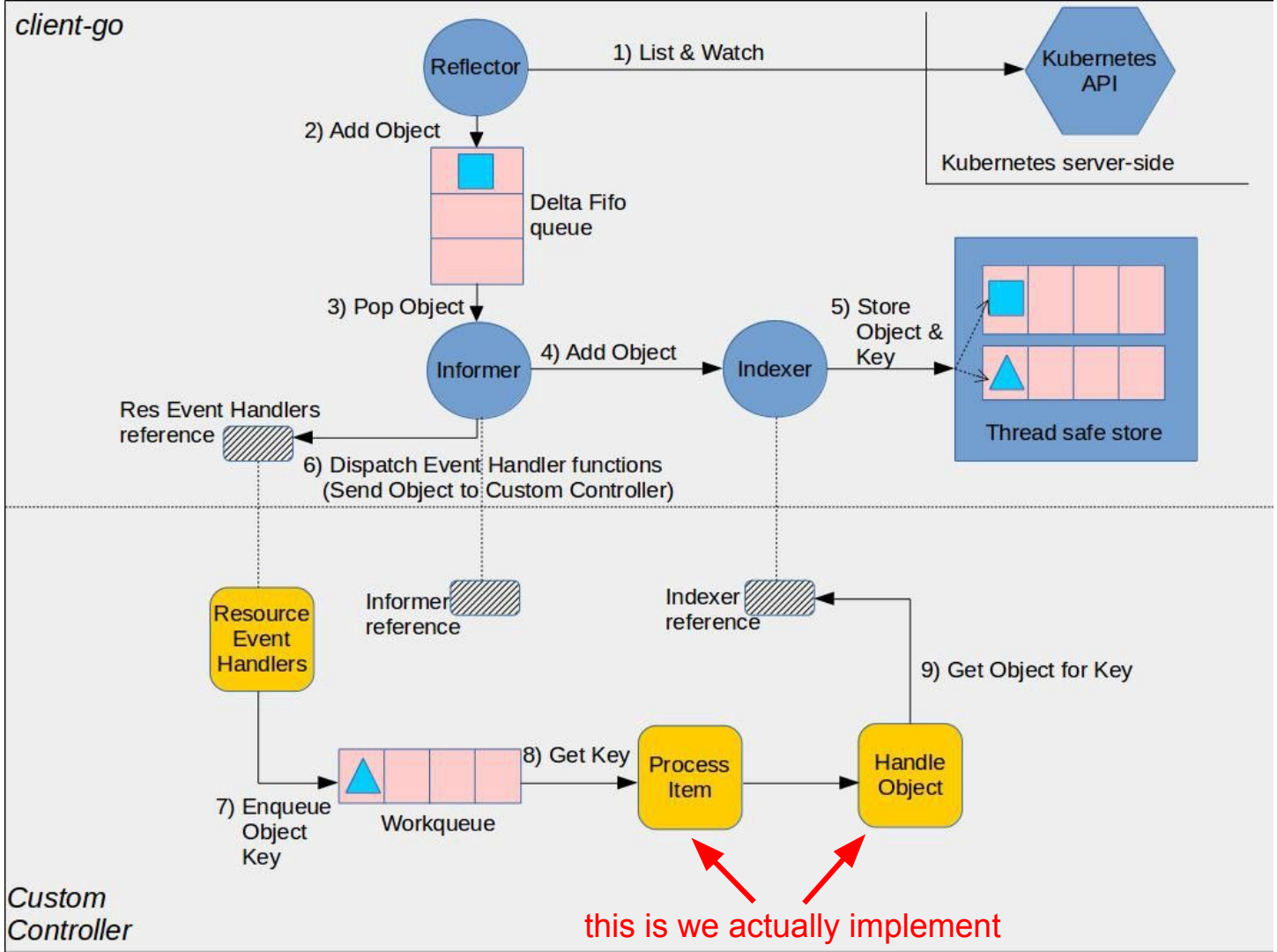


Image by @cloudark / Devdatta Kulkarni
https://medium.com/@cloudark/kubernetes-custom-controllers-b6c7d0668fdf

Inner workings of a Controller

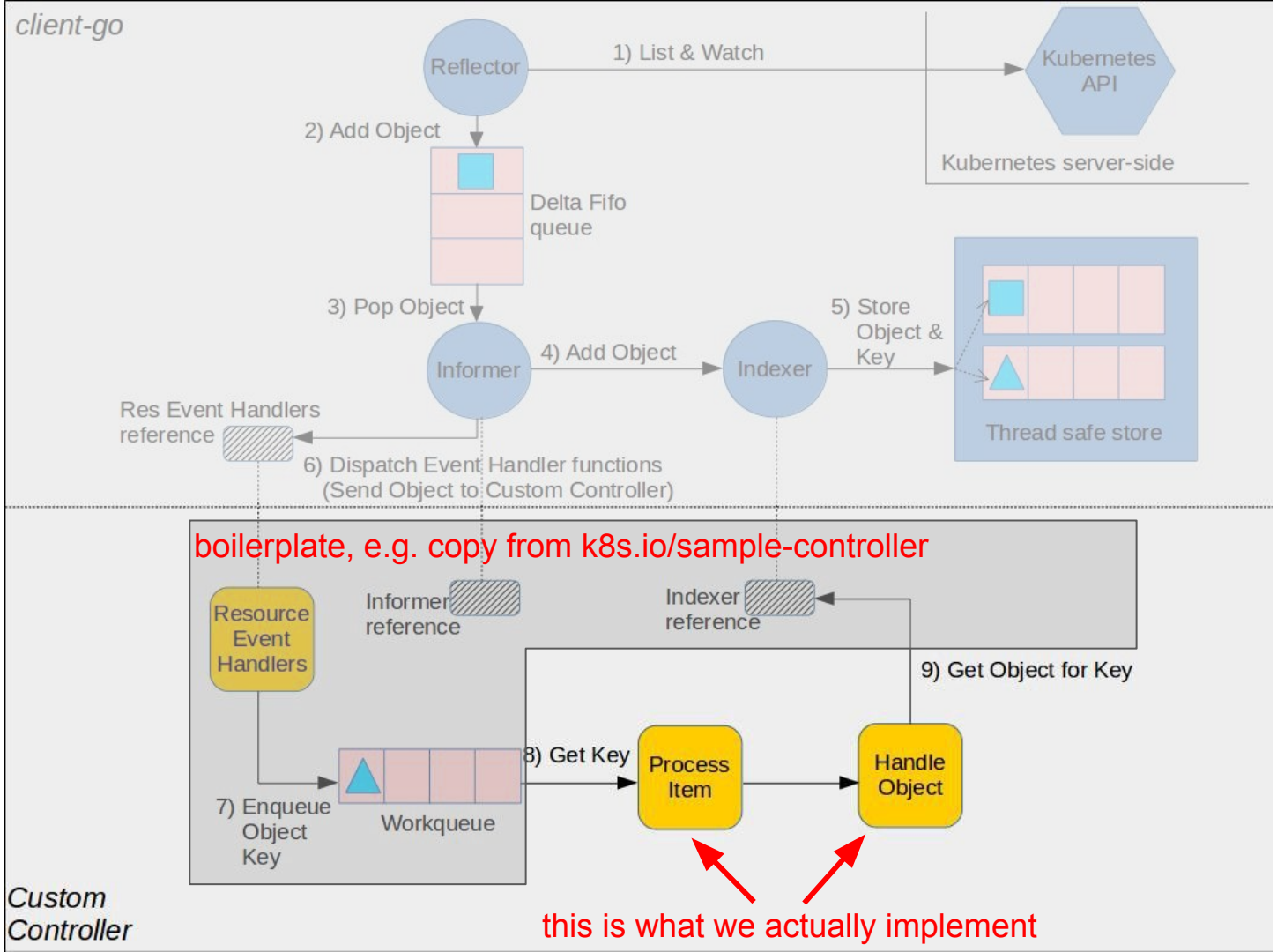


Image by @cloudark / Devdatta Kulkarni
https://medium.com/@cloudark/kubernetes-custom-controllers-b6c7d0668fdf

```
$ while true; do
        http GET http://127.0.0.1:8080/api/v1/namespaces/default |
                {jq ".metadata.annotations[\"example\"] = \"$RANDOM\""; sleep 1;} |
                http --check-status PUT http://127.0.0.1:8080/api/v1/namespaces/default || break
  done
```

„optimistic concurrency“

```
HTTP/1.1 409 Conflict
Content-Length: 310
Content-Type: application/json
Date: Fri, 10 Mar 2017 08:27:58 GMT
{
        "apiVersion": "v1",
        "code": 409,
        "details": {
                "kind": "namespaces",
                "name": "default“
        },
        "kind": "Status",
        "message": "Operation cannot be fulfilled on namespaces \"default\": the object has been
                        modified; please apply your changes to the latest version and try again",
        "metadata": {},
        "reason": "Conflict",
        "status": "Failure“
}
```

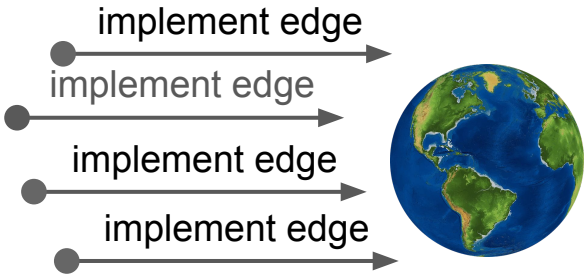
1. **read** object (preferably event driven with **watches**)
2. **change** object, update the world
3. **update** object on apiserver
4. **repeat**

expect version conflicts,
remember: **optimistic concurrency**

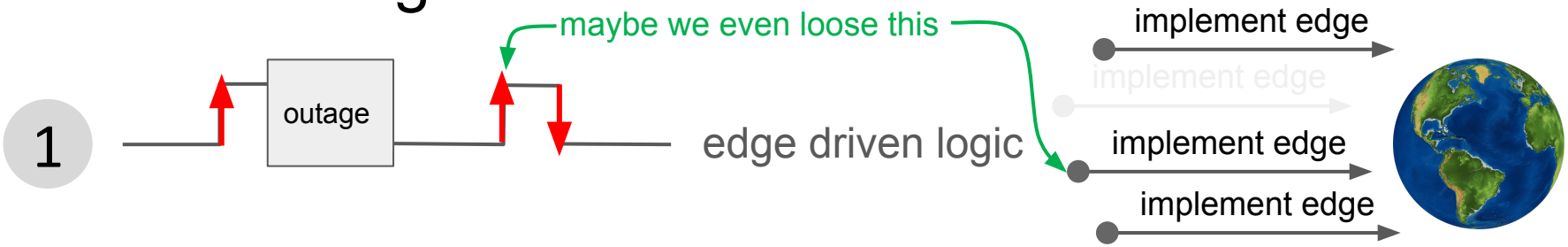
Level vs. Edge driven

1

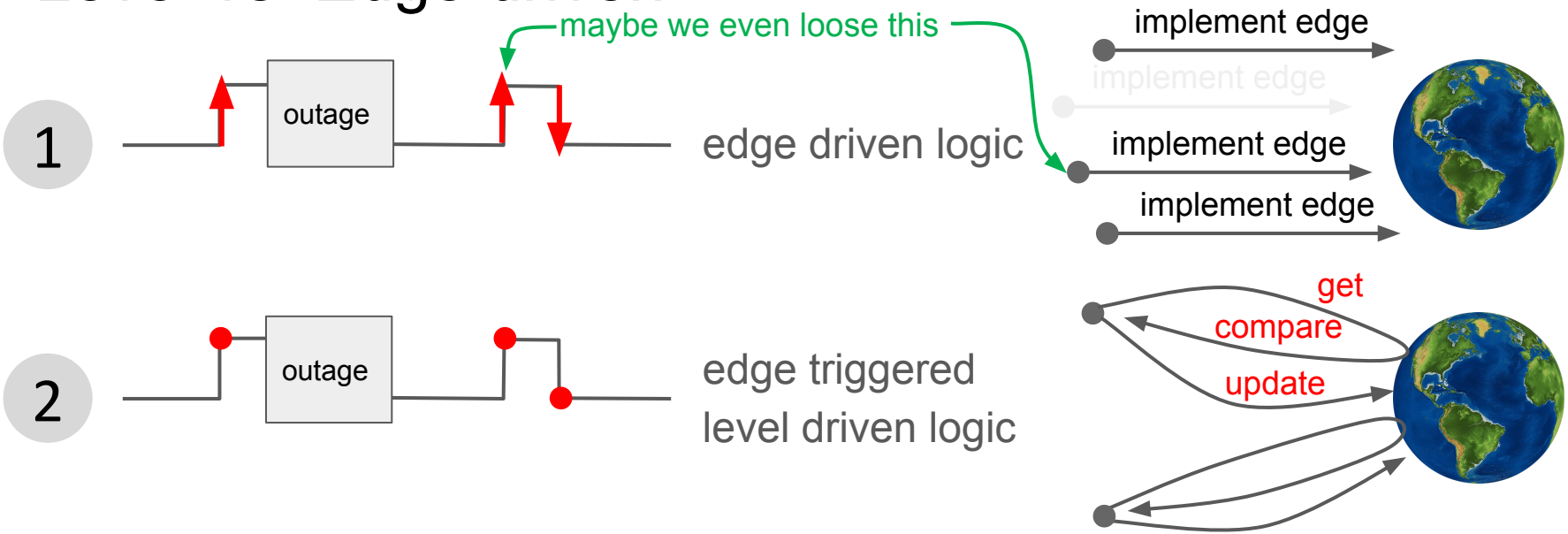
edge driven logic



Level vs. Edge driven

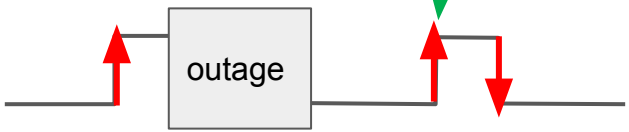


Level vs. Edge driven



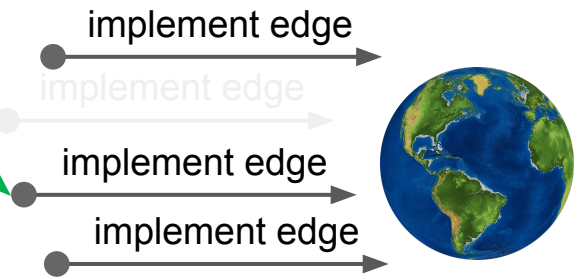
Level vs. Edge driven

1



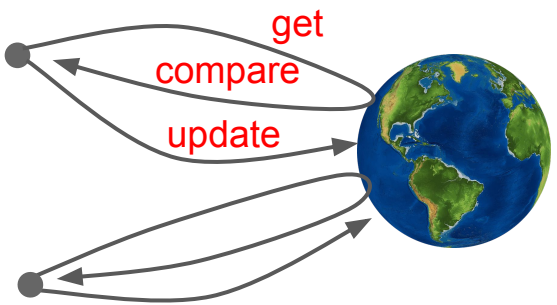
edge driven logic

maybe we even loose this

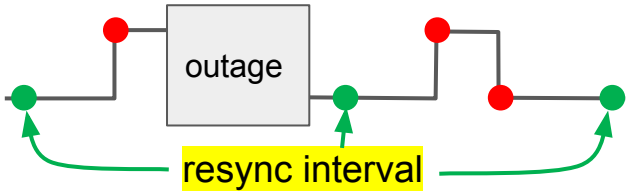


2

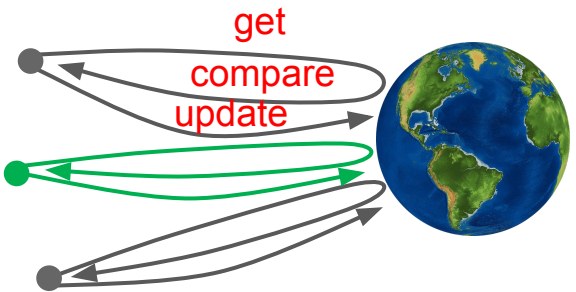
edge triggered
level driven logic



3



reconciliation
with resync



```go
func (c *Controller) syncHandler(key string) error {
    // Convert the namespace/name string into a distinct namespace and name
    namespace, name, err := cache.SplitMetaNamespaceKey(key)
    if err != nil {
        runtime.HandleError(fmt.Errorf( format: "invalid resource key: %s", key))
        return nil
    }

    // Get the health check policy resource with this namespace/name
    policy, err := c.healthCheckPolicyLister.HealthCheckPolicies(namespace).Get(name)
    if err != nil {
        if errors.IsNotFound(err) {
            runtime.HandleError(fmt.Errorf( format: "health check policy '%s' in work queue no longer exists", key))
            return nil
        }

        return err
    }

    // keep original, and deep copy policy before mutating it
    original := policy
    policy = policy.DeepCopy()

    // ########################################
    // # TODO: add controller logic here
    // ########################################
    policy.Status.PodsFailed = 42

    // skip update if there was no change
    if reflect.DeepEqual(original, policy) {
        return nil
    }

    // there was a change. Update resource on the server.
    if _, err := c.policyClient.PolicyV1alpha1().HealthCheckPolicies(policy.Namespace).Update(policy); errors.IsConflict(err) {
        c.workqueue.Add(key)
        return err
    } else if err != nil {
        return err
    }

    // no need to requeue, we will see our own update.

    return nil
}
```

github.com/openshift/k8s-go/controller/cmd/policy-controller/controller/controller.go

split key into namespace+name

get object from cache

keep copy
deepcopy before mutation

update in memory

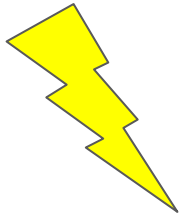
anything changed?

update on server

requeue on conflict

References for controllers

- k8s.io/sample-controller
- Maciej Szulik's talk from KubeCon EU 2018
- Writing Kubernetes Custom Controllers by Devdatta Kulkarni
- the controllers in core: k8s.io/kubernetes/pkg/controller



Katacoda

→ Writing controllers (step 6)

kubebuilder

kubebuilder init --domain k8s-go.openshift.org
--license apache2
--owner "The workshop members"

```
Run `dep ensure` to fetch dependencies (Recommended) [y/n]?
y
dep ensure
Running make...
make
go generate ./pkg/... ./cmd/...
go fmt ./pkg/... ./cmd/...
go vet ./pkg/... ./cmd/...
go run vendor/sigs.k8s.io/controller-tools/cmd/controller-gen/main.go all
CRD manifests generated under '/Users/sts/Quellen/kubernetes/src/github.com/openshift/k8s-go/kubebuilder/config/crds'
RBAC manifests generated under '/Users/sts/Quellen/kubernetes/src/github.com/openshift/k8s-go/kubebuilder/config/rbac'
go test ./pkg/... ./cmd/... -coverprofile cover.out
?       github.com/openshift/k8s-go/kubebuilder/pkg/apis        [no test files]
?       github.com/openshift/k8s-go/kubebuilder/pkg/controller  [no test files]
?       github.com/openshift/k8s-go/kubebuilder/cmd/manager     [no test files]
go build -o bin/manager github.com/openshift/k8s-go/kubebuilder/cmd/manager
Next: Define a resource with:
$ kubebuilder create api
```

kubebuilder create api --group policy*
--version v1beta1
--kind HealthCheckPolicy

```
Create Resource under pkg/apis [y/n]?
y
Create Controller under pkg/controller [y/n]?
y
Writing scaffold for you to edit...
pkg/apis/policy/v1beta1/healthcheckpolicy_types.go
pkg/apis/policy/v1beta1/healthcheckpolicy_types_test.go
pkg/controller/healthcheckpolicy/healthcheckpolicy_controller.go
pkg/controller/healthcheckpolicy/healthcheckpolicy_controller_test.go
Running make...
go generate ./pkg/... ./cmd/...
```

* needs some fixing of the controller code due to wrong import: https://github.com/kubernetes-sigs/kubebuilder/issues/335

https://github.com/openshift-talks/k8s-go/blob/master/kubebuilder/main.go

```
import (
    kubescheme "k8s.io/client-go/kubernetes/scheme"
    kubebuilderclient "sigs.k8s.io/controller-runtime/pkg/client"
)

kubeconfig = flag.String("kubeconfig", "~/.kube/config", "path to the kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)

client, err := kubebuilderclient.NewForConfig(config, Options{Scheme: kubescheme.Scheme})
pod := corev1.Pod{}
err := client.Get(context.TODO(), {"gophercon", "workshop"}, &pod)
```

as usual

Get, Create, Update, List, Delete, ~~Patch~~, ~~Watch~~

uses API discovery to know HTTP paths

```go
func (r *ReconcileHealthCheckPolicy) Reconcile(request reconcile.Request) (reconcile.Result, error) {
    // Fetch the HealthCheckPolicy instance
    instance := &policyv1beta1.HealthCheckPolicy{}
    err := r.Get(context.TODO(), request.NamespacedName, instance)
    if err != nil {
        if errors.IsNotFound(err) {
            // Object not found, return.  Created objects are automatically garbage collected.
            // For additional cleanup logic use finalizers.
            return reconcile.Result{}, nil
        }
        // Error reading the object - requeue the request.
        return reconcile.Result{}, err
    }

    // TODO(user): Change this for the object type created by your controller
    // Update the found object and write the result back if there are any changes
    if !reflect.DeepEqual(deploy.Spec, found.Spec) {
        found.Spec = deploy.Spec
        log.Printf( format: "Updating Deployment %s/%s\n", deploy.Namespace, deploy.Name)
        err = r.Update(context.TODO(), found)
        if err != nil {
            return reconcile.Result{}, err
        }
    }
    return reconcile.Result{}, nil
}
```

Namespace + Name

github.com/openshift/k8s-go/kubebuilder/pkg/controller/healthcheckpolicy/healthcheckpolicy_controller.go

The Operator SDK

operator-sdk new app-operator

--api-version=policy.k8s-go.openshift.org/v1beta1

--kind=HealthCheckPolicy

```
Create app-operator/.gitignore
Create app-operator/cmd/app-operator/main.go
Create app-operator/config/config.yaml
Create app-operator/deploy/rbac.yaml
Create app-operator/deploy/crd.yaml
Create app-operator/deploy/cr.yaml
Create app-operator/pkg/apis/policy/v1beta1/doc.go
Create app-operator/pkg/apis/policy/v1beta1/register.go
Create app-operator/pkg/apis/policy/v1beta1/types.go
Create app-operator/pkg/stub/handler.go
Create app-operator/tmp/build/build.sh
Create app-operator/tmp/build/docker_build.sh
Create app-operator/tmp/build/Dockerfile
Create app-operator/tmp/codegen/boilerplate.go.txt
Create app-operator/tmp/codegen/update-generated.sh
Create app-operator/version/version.go
Create app-operator/Gopkg.toml
Run dep ensure ...
```

operator-sdk – workflow

$ **operator-sdk build** <docker-image-name>

$ **docker push** <docker-image-name>

$ OPERATOR_NAME=our-operator **operator-sdk up local**

```
INFO[0000] Go Version: go1.10.3
INFO[0000] Go OS/Arch: darwin/amd64
INFO[0000] operator-sdk Version: 0.0.5+git

INFO[0003] Metrics service foo created
INFO[0003] Watching policy.k8s-go.openshift.org/v1beta1, HealthCheckPolicy, default, 5
```

```
import (
        "github.com/operator-framework/operator-sdk/pkg/sdk"
        corev1 "k8s.io/api/core/v1"
)

pod := corev1.Pod{
        TypeMeta: {ApiVersion: "v1", Kind: "Pod"},
        ObjectMeta: {Name: "busybox", Namespace: "default"},
        Spec: ...
}
err := sdk.Create(&pod)
```

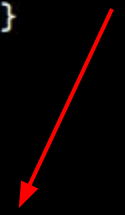
must be filled in

uses API discovery to know HTTP paths

Get, Create, Update, List, Delete, ~~Patch~~, Watch

```go
func NewHandler() sdk.Handler {
    return &Handler{}
}

type Handler struct {
    // Fill me
}

func (h *Handler) Handle(ctx context.Context, event sdk.Event) error {
    switch o := event.Object.(type) {
    case *v1beta1.HealthCheckPolicy:
        err := sdk.Create(newbusyBoxPod(o))
        if err != nil && !errors.IsAlreadyExists(err) {
            logrus.Errorf( format: "Failed to create busybox pod : %v", err)
            return err
        }
    }
    return nil
}
```

```go
type Event struct {
    Object  Object
    Deleted bool
}
```



Client call

github.com/openshift/k8s-go/operator-sdk/app-operator/pkg/stub/handler.go

References for operator-sdk

https://github.com/operator-framework/operator-sdk

A complete guide to Kubernetes Operator SDK by Toader Sebastian

Building an Kubernetes Operator for Prometheus and Thanos by Rob Szumski

Agenda

- **API basics:** apigroups, resource vs. kind, versioning, discovery, **curl scenario**
- **Client-go basics:** clientsets, kubeconfig, **client-go scenario**, informers, **informer scenario**
- **Kubernetes objects in Golang:** ObjectMeta, TypeMeta
- **API Machinery:** Scheme, GVK, GVR, RestMapper
- **CRDs from user point of view:** CRD yaml, Validation, **CRD yaml scenario**
- **Code-Generation:** deepcopy-gen, client-gen, informer-gen, lister-gen
- **CRDs from developer point of view:** types.go, register.go, pkg/apis, code-generator, **CRD Go scenario**
- **Controllers:** controller loop, optimistic concurrency, edge-vs-level driven, **controller scenario**
- **Kubebuilder:** init, create api, **kubebuilder scenario – trainer led**
- **Operator SDK:** new operator, **operator scenario – trainer led**
- **Other topics:** deployment / in-cluster config, RBAC rules, service account, subresources, versioning, **misc scenario ("deploy your controller in the cluster")**

Quick glimpse on other Topics

Deployment vs. in-cluster config

```
import (
     "k8s.io/client-go/tools/clientcmd"
)
kubeconfig = flag.String("kubeconfig", "~/.kube/config", "path to the kubeconfig file")
flag.Parse()
```

looks at /var/run/secrets/kubernetes.io/serviceaccount/token and consider:

- $KUBERNETES_SERVICE_HOST
- $KUBERNETES_SERVICE_PORT

```
config, err := rest.InClusterConfig()
if err != nil {
     config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
     if err != nil { … }
}

clientset, err := kubernetes.NewForConfig(config)
```

client-go dynamic client – unstructured.Unstructed

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/tools/clientcmd"
    "k8s.io/client-go/dynamic"
)

kubeconfig = flag.String("kubeconfig", "~/.kube/config", "path to the kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
client, err := dynamic.NewForConfig(config)

gvr := runtime.GroupVersionResource{Group: "", Version: "v1", Resource: "pods"}
pod, err := client.Resource(gvr).Namespace("gophercon").Get("workshop", {})
```

*unstructured.Unstructed

Get, Create, Update, List, Delete, Patch, Watch

RBAC rules

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: policy-controller-role
rules:
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - policy.k8s-go.openshift.org
  resources:
  - healthcheckpolicies
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: policy-controller-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: policy-controller-role
subjects:
- namespace: controller-namespace
  kind: ServiceAccount
  name: default
```

User or Group or ServiceAccount

every namespace has a ServiceAccount named `default`

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Custom Service Account

```
$ kubectl create serviceaccount policy-controller
```

```
apiVersion: v1
kind: Pod
metadata:
  name: policy-controller
spec:
  serviceAccountName: policy-controller
  ....
```

https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

Subresources – /scale

$ **kubectl scale** --replicas=3 <your-custom-resource> --v=7
I0429 21:17:53.138353 66743 round_trippers.go:383] PUT

https://<host>/apis/<group>/v1/<your-custom-resource>/scale

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
spec:
  subresources:
    scale:
      specReplicasPath: .spec.replicas
      statusReplicasPath: .status.replicas
      labelSelectorPath: .status.labelSelector
```

Subresources – /status

- **separate RBAC rules** for controller and users
  - controller writes to `/apis/<group>/v1/<resource>/status`
  - user writes to `/apis/<group>/v1/<resource>`

  ⇒ The user cannot mutate the status.

- on `/status` all fields outside of `.status` are ignored
- on `/` all fields under `.status` are ignored

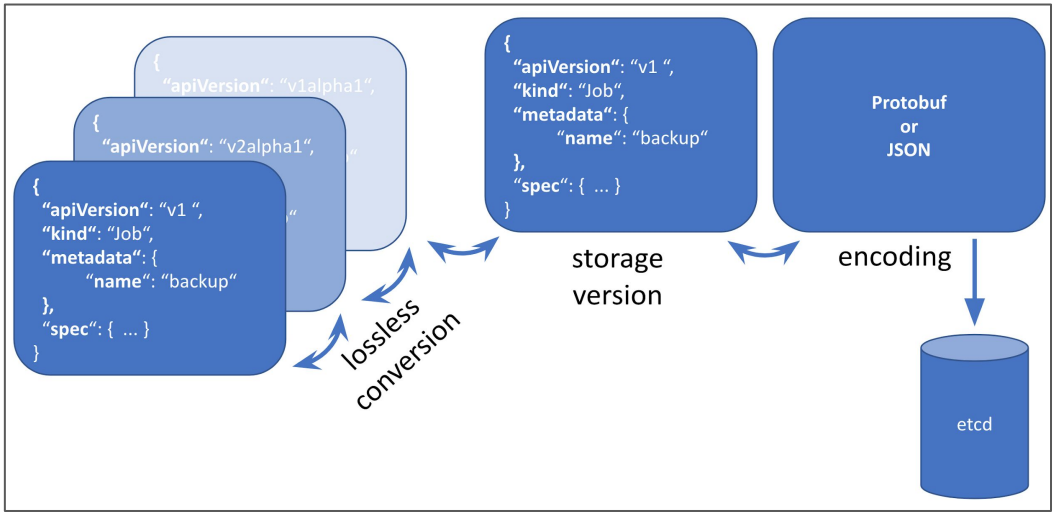
```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
spec:
  subresources:
    status: {}
```


enabled like this, available in Kubernetes 1.11 as beta

Kubernetes Style Versioning

```
v1alpha1
v1alpha2
v1beta1
v1
v2alpha1
v2beta1
v2
```

- v1alpha1 – unstable:
  - might go away or change any time
  - often disabled by default
- v1beta1 – towards stable:
  - exists at least one release in parallel to v1
  - no incompatible API change
- v1 – stable, GA:
  - will stay
  - will be compatible



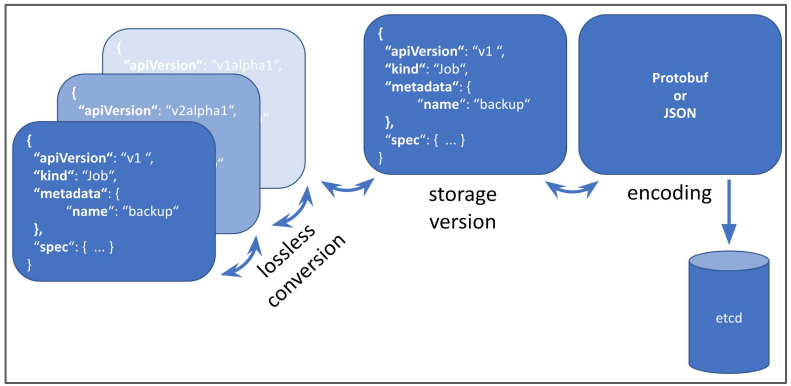
CRD Versions in Kubernetes 1.11 and beyond

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
spec:
   version: v1alpha1
   versions:
   - name: v1alpha1
     served: true
     storage: false
   - name: v1beta1
     served: true
     storage: true
   conversion:
     webhook:
       clientConfig:
         url: ...
```

will be slowly deprecated

in Kubernetes 1.11

planned in Kubernetes 1.12 as alpha



Agenda

- **API basics:** apigroups, resource vs. kind, versioning, discovery, **curl scenario**
- **Client-go basics:** clientsets, kubeconfig, **client-go scenario**, informers, **informer scenario**
- **Kubernetes objects in Golang:** ObjectMeta, TypeMeta
- **API Machinery:** Scheme, GVK, GVR, RestMapper
- **CRDs from user point of view:** CRD yaml, Validation, **CRD yaml scenario**
- **Code-Generation:** deepcopy-gen, client-gen, informer-gen, lister-gen
- **CRDs from developer point of view:** types.go, register.go, pkg/apis, code-generator, **CRD Go scenario**
- **Controllers:** controller loop, optimistic concurrency, edge-vs-level driven, **controller scenario**
- **Kubebuilder:** init, create api, **kubebuilder scenario – trainer led**
- **Operator SDK:** new operator, **operator scenario – trainer led**
- **Other topics:** deployment / in-cluster config, RBAC rules, service account, subresources, versioning, **misc scenario ("deploy your controller in the cluster")**

Backup

Zoom into apiextensions-apiserver