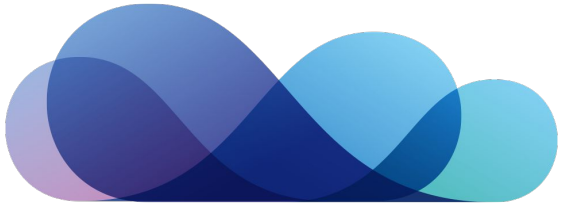


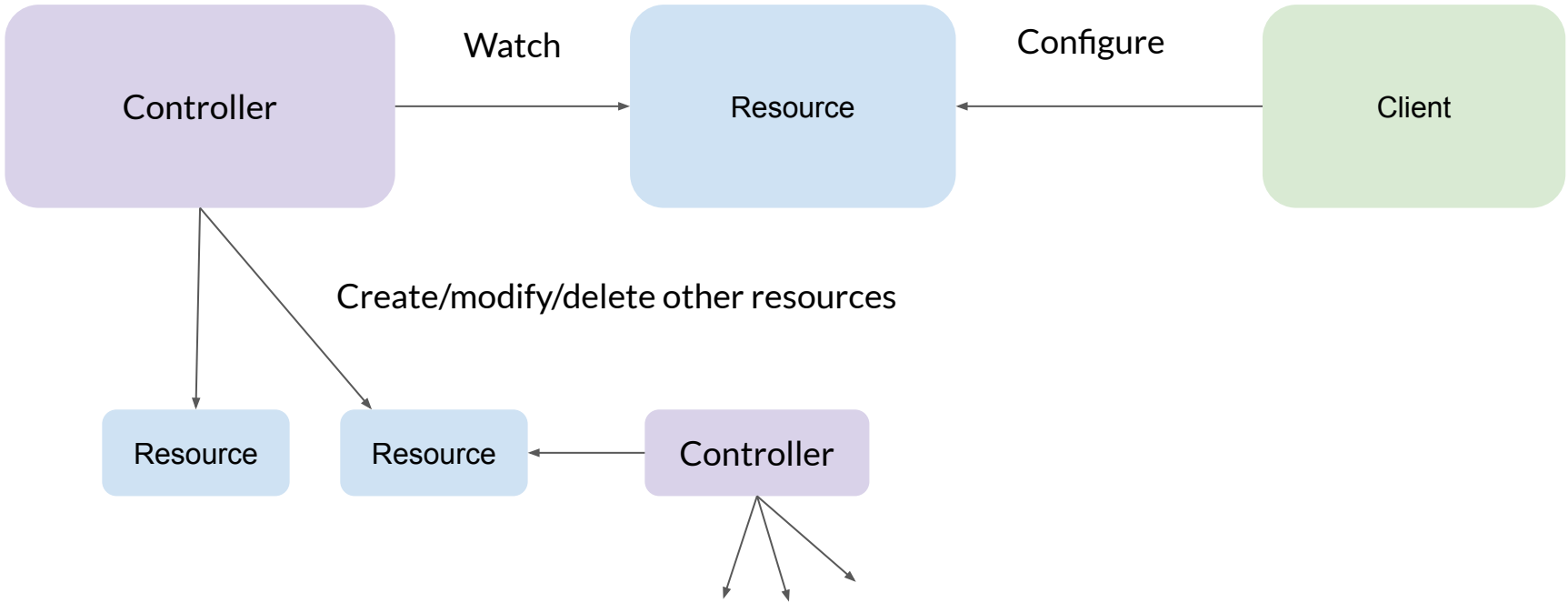

# Reconciliation on different levels

Peter Wilcsinszky (pepov)

banzaicloud.com

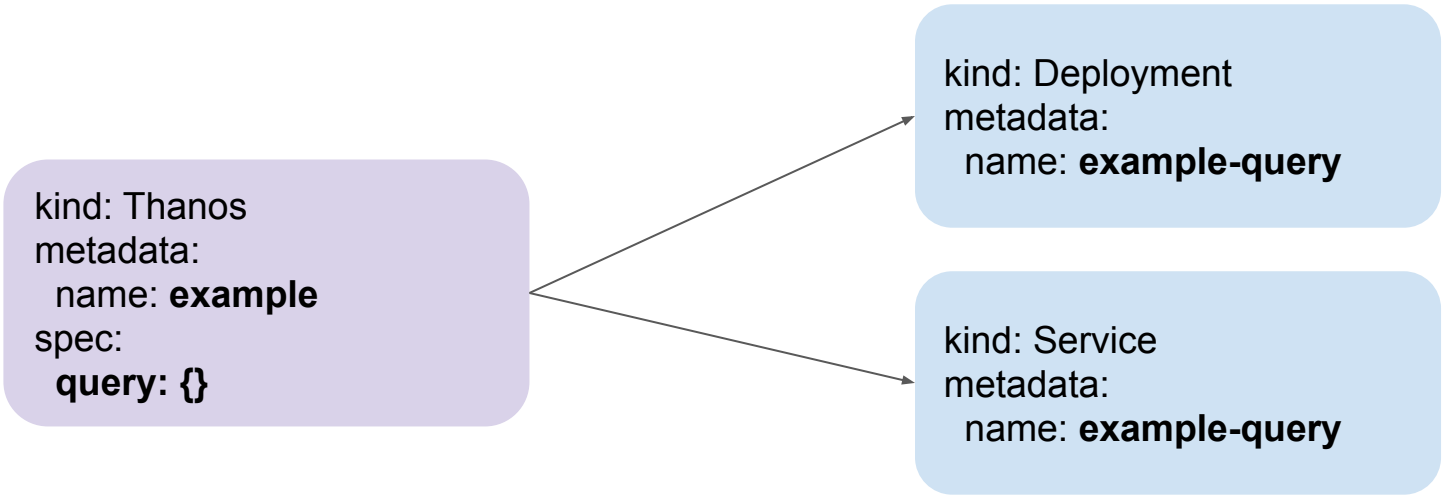
# Reconcile

# Reconcile

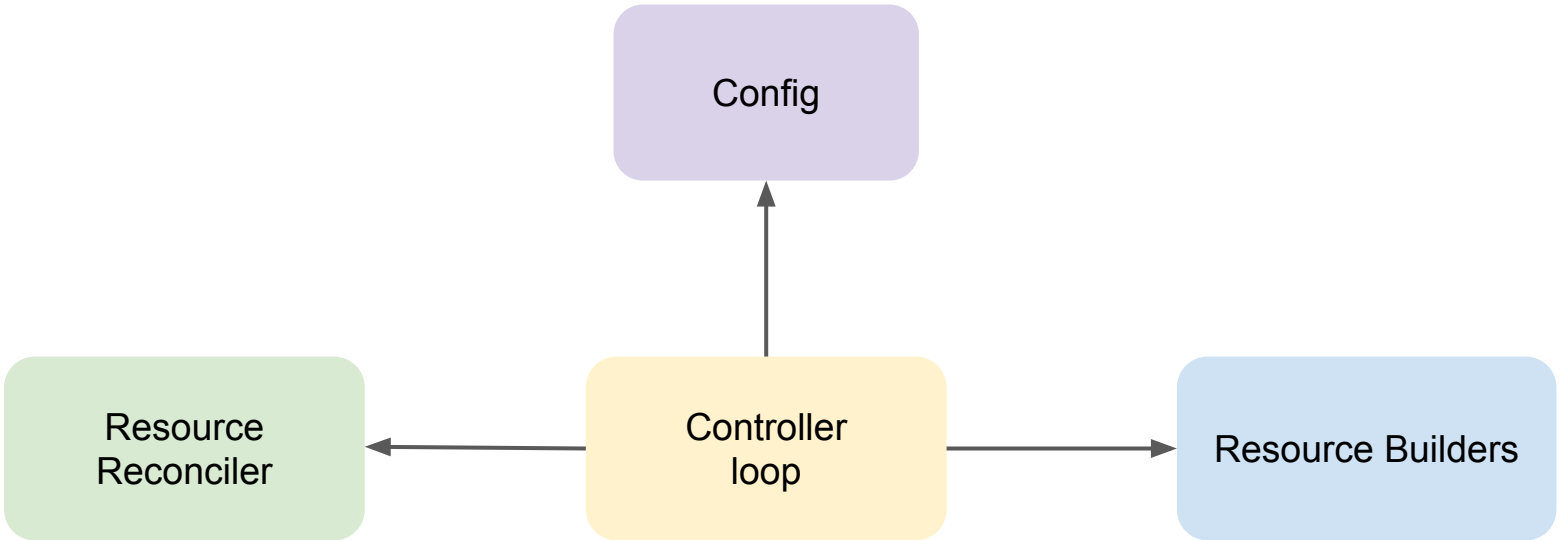


https://github.com/banzaicloud/thanos-operator

```
apiVersion: monitoring.banzaicloud.io/v1alpha1
kind: Thanos
metadata:
  name: exampe
spec:
  query: {}
```




Deterministic names instead of storing IDs



https://github.com/banzaicloud/operator-tools/tree/master/pkg/reconciler

```
type ResourceBuilder func(runtime.Object) (runtime.Object, DesiredState)

type ResourceReconciler interface {
    ReconcileResource(runtime.Object, DesiredState) (*reconcile.Result, error)
}
```

```
type StaticDesiredState string

const (
  StateAbsent  StaticDesiredState = "Absent"
  StatePresent StaticDesiredState = "Present"
)

type DesiredState interface {
    BeforeUpdate(current runtime.Object) error
}
```

How do we know we have to update a resource?

Can’t we update it all the time?

Kubernetes API server would handle these updates, but…

There would be lots of unnecessary connections to the API server

We lose the visibility into what we are doing exactly, i.e. the ability to track down inconsistencies

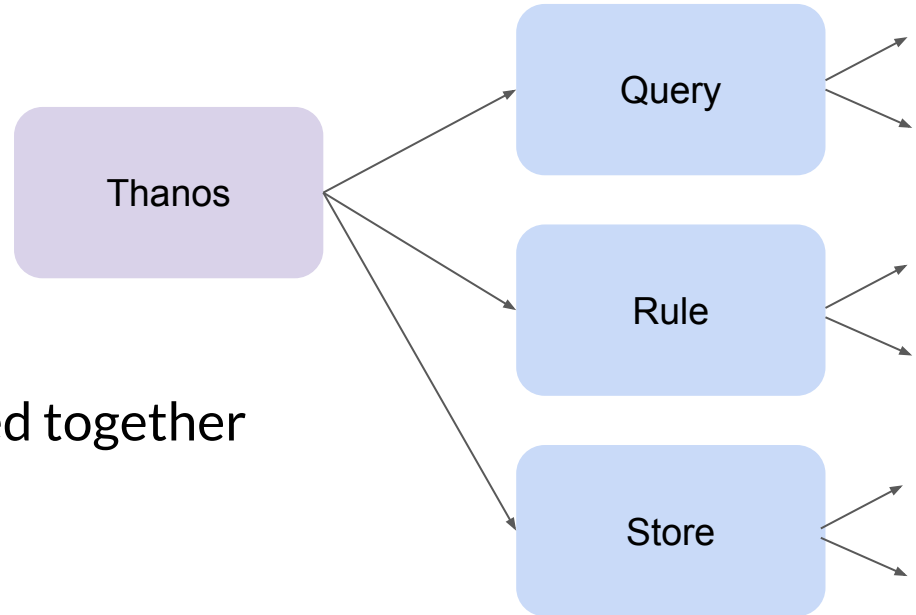
Get the existing resource (from the cache)

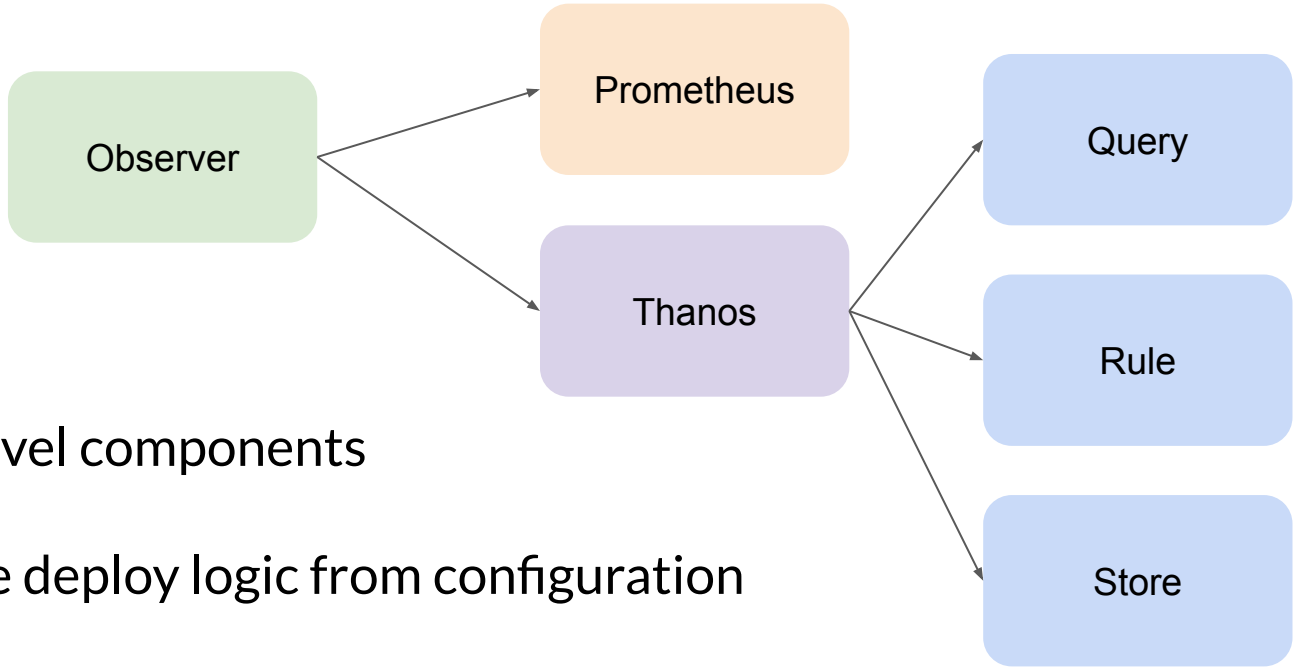
Compare with the original version using
https://github.com/banzaicloud/k8s-objectmatcher

Update only if there is a diff

Component

A bunch of resources managed together






Higher level components

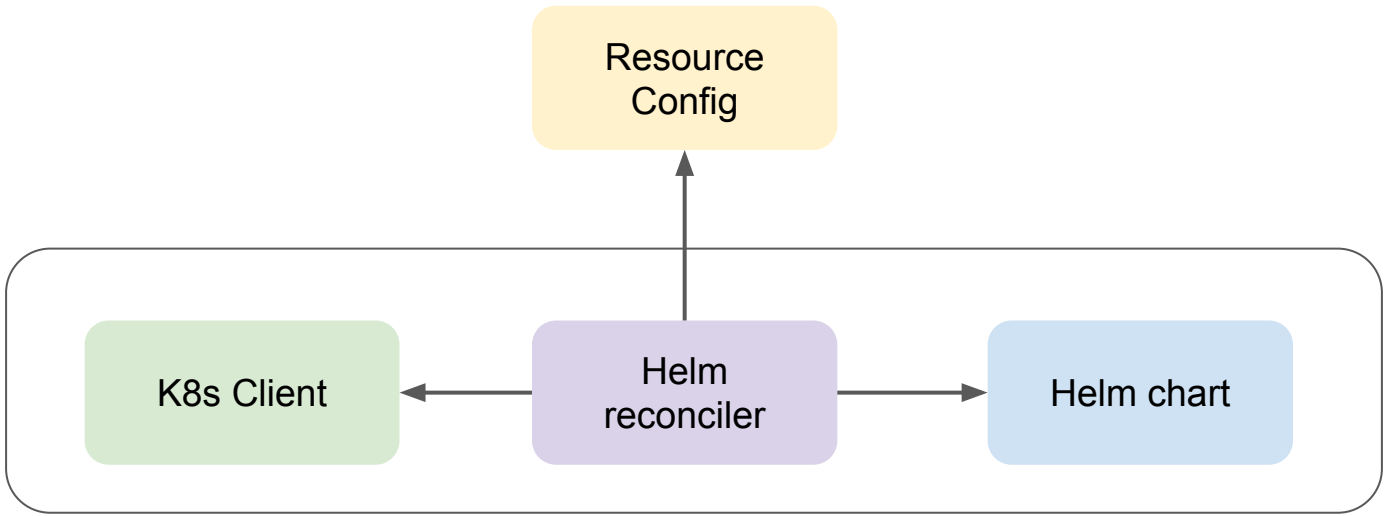
Decouple deploy logic from configuration

## Possible implementations

- Helm v3
- External resource builder
- `whatnot | kubectl apply -f-`

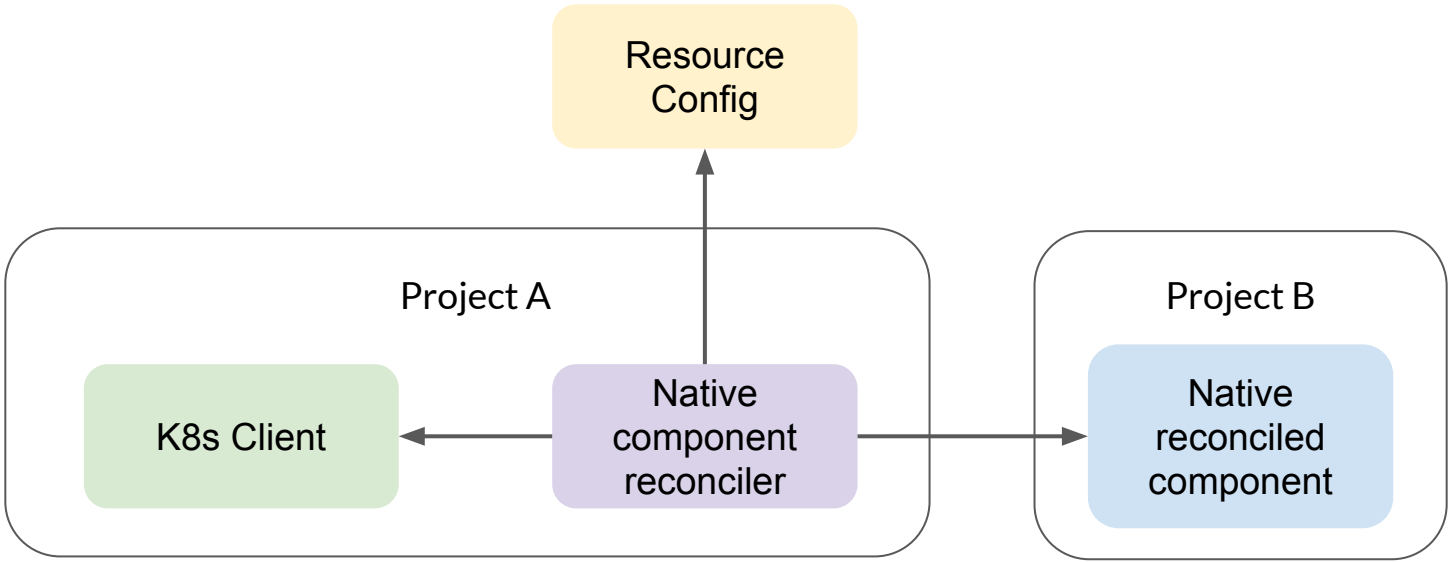
Pro: use existing Helm charts

Con: little to no control over the configuration



Pro: tight control over resources, flexible object manipulation

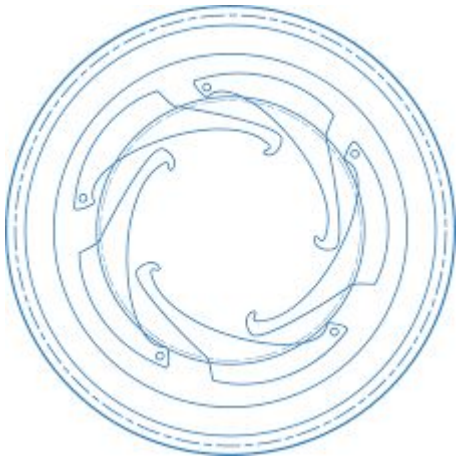
Con: need to write code for it

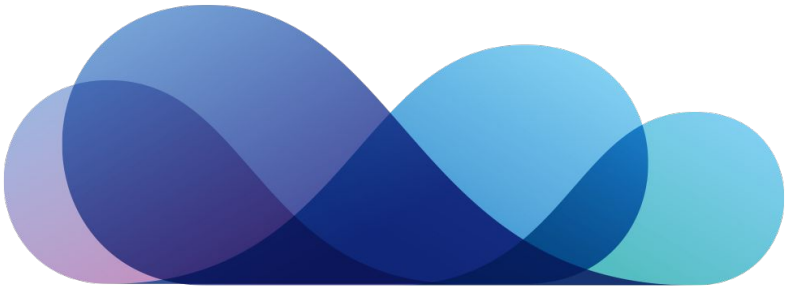


```go
func ResourceBuilders(parent reconciler.ResourceOwner, object interface{}) []reconciler.ResourceBuilder {
	config := &ComponentConfig{
		Disabled: true,
	}
	if object != nil {
		config = object.(*ComponentConfig)
	}
	resources := []reconciler.ResourceBuilder{
		config.build(parent, Operator),
		config.build(parent, ClusterRole),
		config.build(parent, ClusterRoleBinding),
		config.build(parent, ServiceAccount),
	}
	// We don't return with an absent state since we don't want them to be removed
	if !config.Disabled {
		resources = append(resources,
			func() (runtime.Object, reconciler.DesiredState, error) {
				return CRD(config, v1alpha1.GroupVersion.Group, "objectstores")
			},
			func() (runtime.Object, reconciler.DesiredState, error) {
				return CRD(config, v1alpha1.GroupVersion.Group, "thanos")
			},
			func() (runtime.Object, reconciler.DesiredState, error) {
				return CRD(config, v1alpha1.GroupVersion.Group, "storeendpoints")
			},
		)
	}
	return resources
}
```

https://banzaicloud.com/products/one-eye/

```
apiVersion: one-eye.banzaicloud.io/v1alpha1
kind: Observer
metadata:
  name: thanos
spec:
  thanos:
    operator: {}
  prometheus:
    prometheusOperatorChart: {}
```





BANZAI CLOUD