Best practices for building Kubernetes Operators

Patryk Wasielewski

About me

- DevOps Consultant at Amazon Web Services (AWS)
- 6 years professional experience as DevOps / SRE / Developer
- Cloud-native enthusiast
- Flesh and Blood TCG player

Agenda

- Controllers, Operators? What are Those?
- Validation & Defaulting
- Finalizers
- Local Clusters and testing units
- Loose thoughts?
- Useful links

Controllers, Operators? What are those?

CRDs

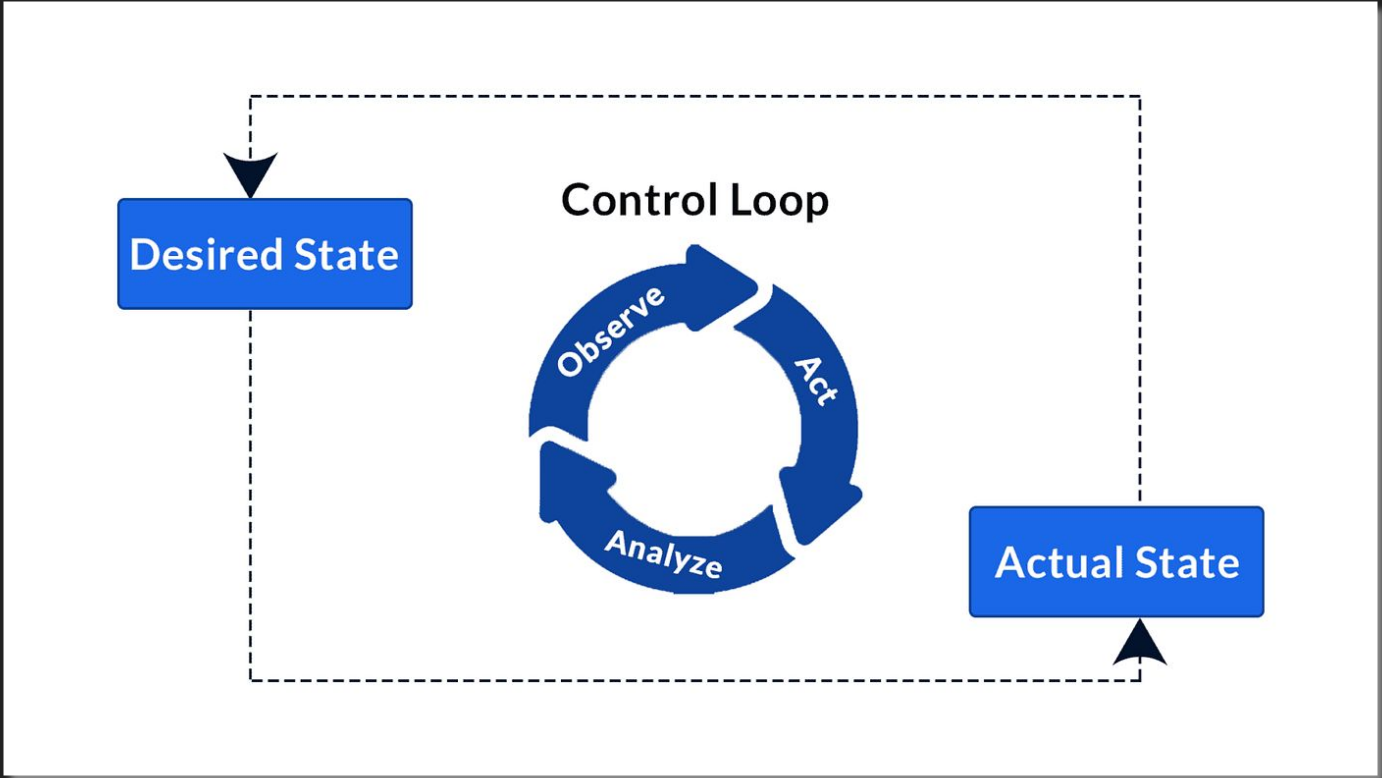
```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties: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

```yaml
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

CRDs

- New RESTful resource path per CRD version
- CRDs/Resources are CRUD
- CR can be namespaced or cluster-scoped
- CRDs are cluster-scoped
- Deleting a namespace with CRs results with cascading deletion of those CRs
- The name of a CRD object must be a valid DNS subdomain name
- CRDs are automatically added with bunch of features
  - CRUD
  - Discovery
  - json-patch/merge-patch support
  - Finalizers
  - Built-in Authz/Authn

Controllers



https://able8.medium.com/kubernetes-controllers-overview-b6ec086c1fb

Controllers

- Controller tracks at least one resource type
- It’s a common approach to manage only one resource type per controller
- Controllers reconciliation loop/control loop make any necessary changes to make the desire state of the resource (based on manifest) the actual one
- Operator’s fundamental
- Built-in controller examples:
  - ReplicaSert
  - Deployment
  - StatefulSet
  - Job

Operators

- Concept was introduced in 2016 by the CoreOS
- Operators are software extensions that use custom resources to manage applications and their components
- Using Operators enables us to view an application as a single object that exposes only the adjustments that make sense for the application, instead of a collection of primitives (such as Pods, Deployments, Services, or ConfigMaps).
- Operators actually allow automatic implementation of typical Day-1 tasks (installation, configuration, etc.) and Day-2 tasks (reconfiguration, upgrade, backup, failover, recovery, etc.), for a software running within the Kubernetes cluster,

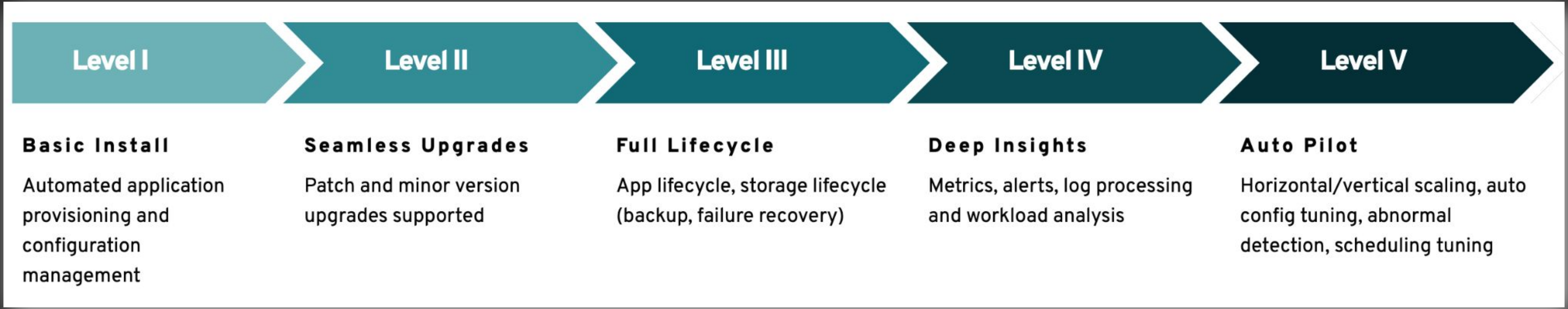
https://www.cncf.io/blog/2022/06/15/kubernetes-operators-what-are-they-some-examples/

Framework - Kubebuilder/Operator-sdk

kubebuilder

OPERATOR SDK

Capability Model



Level I	Level II	Level III	Level IV	Level V
Basic Install	**Seamless Upgrades**	**Full Lifecycle**	**Deep Insights**	**Auto Pilot**
Automated application provisioning and configuration management	Patch and minor version upgrades supported	App lifecycle, storage lifecycle (backup, failure recovery)	Metrics, alerts, log processing and workload analysis	Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

https://operatorframework.io/operator-capabilities/

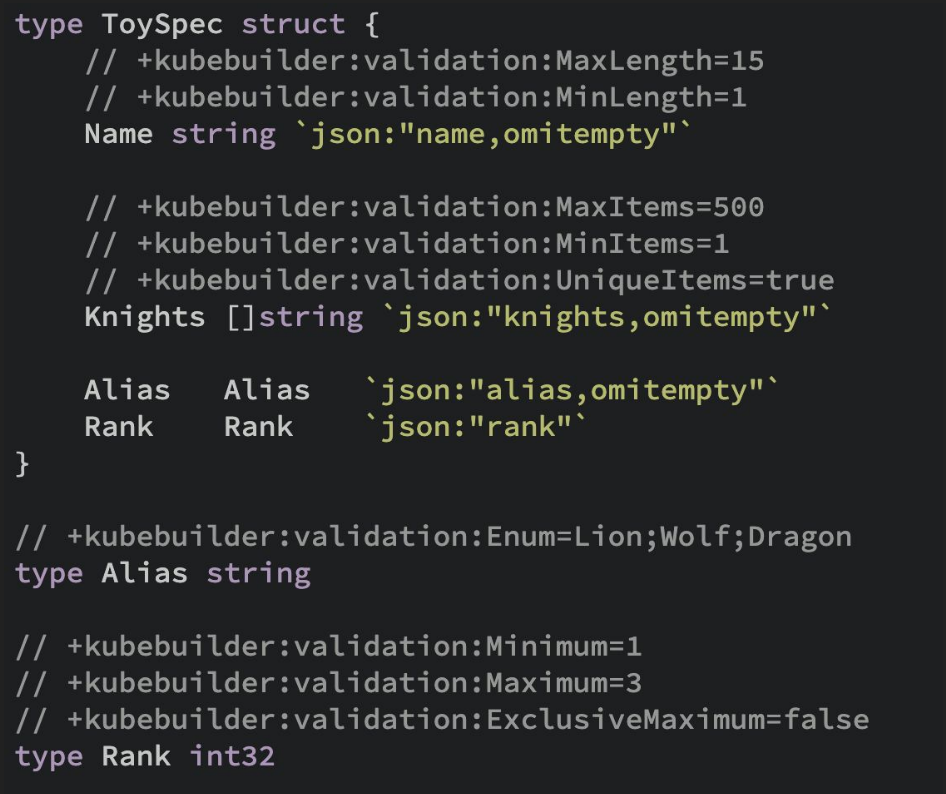
Validation & Defaulting

OpenAPI v3 schemas

- Validation is done on PUT / POST / PATCH requests
- apiVersion, kind, metadata validation
- Value validation
  - maxProperties
  - maxLength
  - enum
  - …
- string formats validation
  - date
  - password
  - byte
  - binary
  - …
- Quantors for subschemas
  - allOf
  - oneOf
  - anyOf
  - not

```yaml
schema:
  openAPIV3Schema:
    type: object
    properties:
      spec:
        type: object
        properties:
          cronSpec:
            type: string
            pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
          finalizers:
            type: array
            items:
              type: string
              pattern: "resource-finalizer"
          image:
            type: string
          replicas:
            type: integer
            minimum: 1
            maximum: 10
```

OpenAPI v3 schemas



```go
type ToySpec struct {
    // +kubebuilder:validation:MaxLength=15
    // +kubebuilder:validation:MinLength=1
    Name string `json:"name,omitempty"`

    // +kubebuilder:validation:MaxItems=500
    // +kubebuilder:validation:MinItems=1
    // +kubebuilder:validation:UniqueItems=true
    Knights []string `json:"knights,omitempty"`

    Alias   Alias   `json:"alias,omitempty"`
    Rank    Rank    `json:"rank"`
}

// +kubebuilder:validation:Enum=Lion;Wolf;Dragon
type Alias string

// +kubebuilder:validation:Minimum=1
// +kubebuilder:validation:Maximum=3
// +kubebuilder:validation:ExclusiveMaximum=false
type Rank int32
```

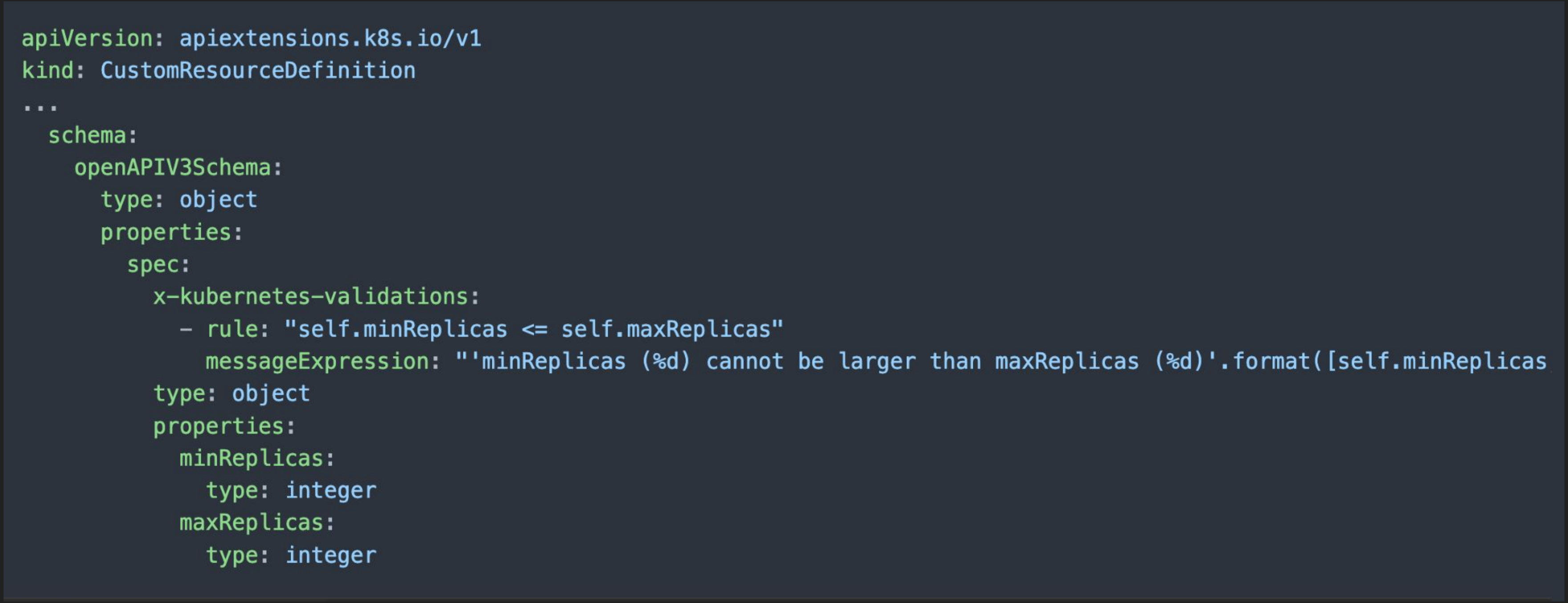
https://book.kubebuilder.io/reference/generating-crd

Validation rules - Common Expression Language (CEL)

- Stable since Kubernetes 1.29
- All validation rules are scoped to the current object
- The rule itself is scoped by the *x-kubernetes-validations location*
- Minimized points of failure in comparison to validating webhooks
- Allows to compare values/sets/objects from the whole manifest
- Custom error messages

https://github.com/kubernetes/enhancements/blob/master/keps/sig-api-machinery/2876-crd-validation-expression-language/README.md

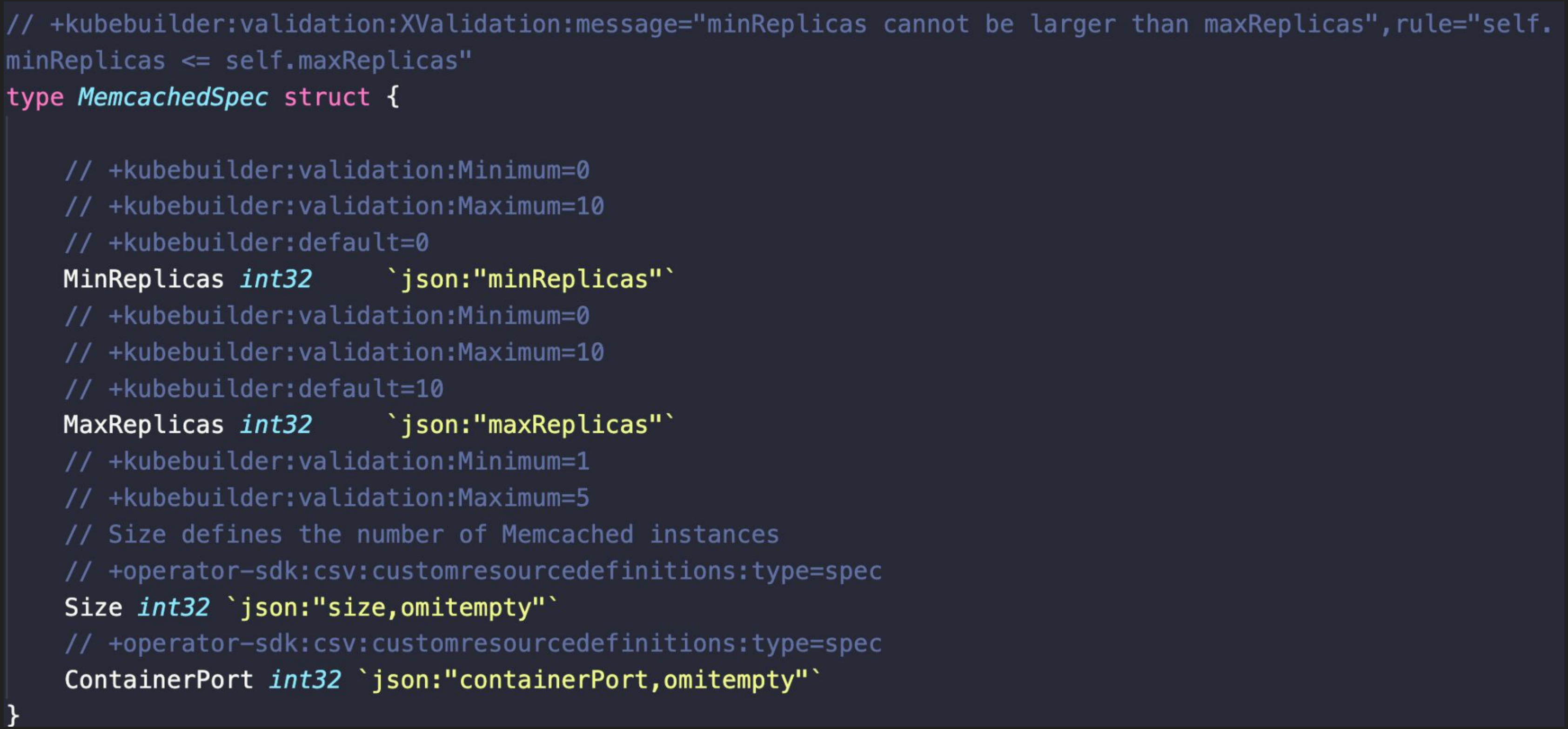
Validation rules - Common Expression Language (CEL)



```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
...
  schema:
    openAPIV3Schema:
      type: object
      properties:
        spec:
          x-kubernetes-validations:
            - rule: "self.minReplicas <= self.maxReplicas"
              messageExpression: "'minReplicas (%d) cannot be larger than maxReplicas (%d)'.format([self.minReplicas
          type: object
          properties:
            minReplicas:
              type: integer
            maxReplicas:
              type: integer
```

https://github.com/kubernetes/enhancements/blob/master/keps/sig-api-machinery/2876-crd-validation-expression-language/README.md

Validation rules - Common Expression Language (CEL)



```go
// +kubebuilder:validation:XValidation:message="minReplicas cannot be larger than maxReplicas",rule="self.
minReplicas <= self.maxReplicas"
type MemcachedSpec struct {

    // +kubebuilder:validation:Minimum=0
    // +kubebuilder:validation:Maximum=10
    // +kubebuilder:default=0
    MinReplicas int32     `json:"minReplicas"`
    // +kubebuilder:validation:Minimum=0
    // +kubebuilder:validation:Maximum=10
    // +kubebuilder:default=10
    MaxReplicas int32     `json:"maxReplicas"`
    // +kubebuilder:validation:Minimum=1
    // +kubebuilder:validation:Maximum=5
    // Size defines the number of Memcached instances
    // +operator-sdk:csv:customresourcedefinitions:type=spec
    Size int32 `json:"size,omitempty"`
    // +operator-sdk:csv:customresourcedefinitions:type=spec
    ContainerPort int32 `json:"containerPort,omitempty"`
}
```

Transition Rules

- To meet the Transition Rules criteria, the object must have be already created
- “A rule that contains an expression referencing the identifier oldSelf is implicitly considered a transition rule”
- Transition rules solves some previously complex cases like:
  - immutability - self.foo == oldSelf.foo
  - Prevent modification/removal once assigned - *oldSelf != 'bar' || self == 'bar' or !has(oldSelf.field) || has(self.field)*
  - setting certain/fixed values, after concrete ones - *oldSelf != 'T' || self in ['A', 'B']*
- Transition Rules shouldn’t be used with optional fields

Transition Rules - Immutable resource

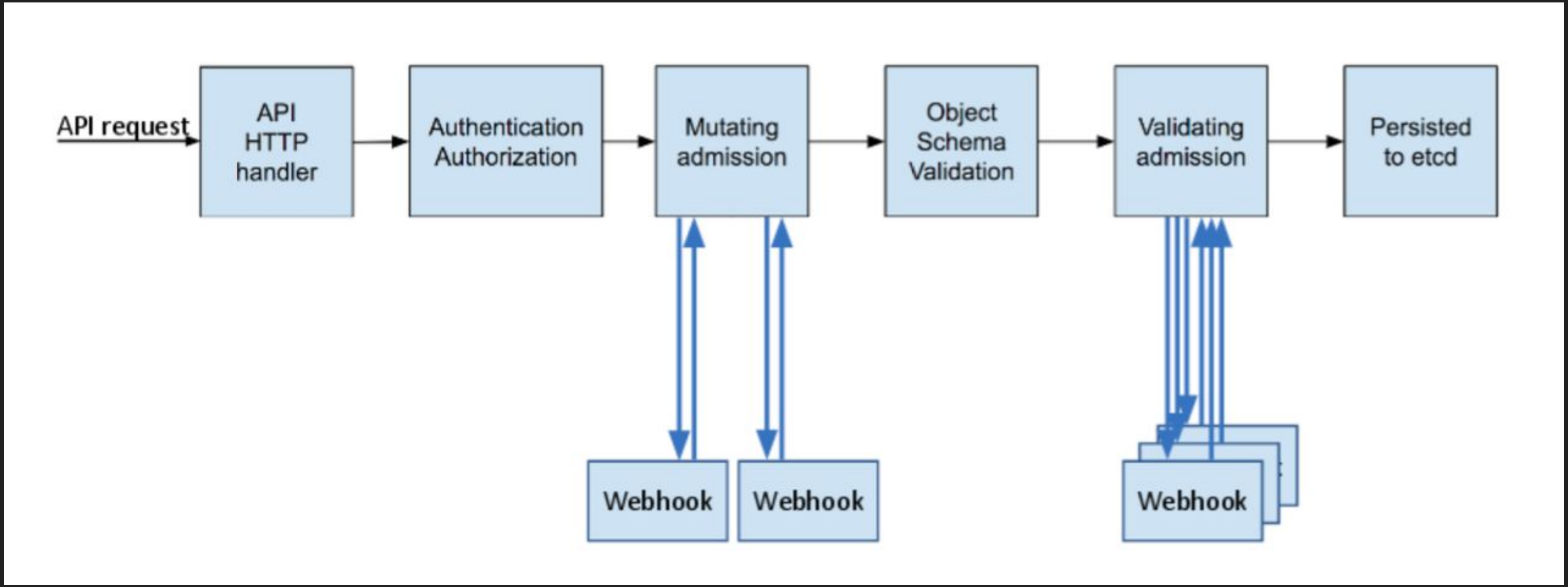
```go
// +kubebuilder:validation:XValidation:rule="self == oldSelf", message="Value is immutable"
type MemcachedSpec struct {
    // +kubebuilder:validation:Minimum=1
    // +kubebuilder:validation:Maximum=5

    // Size defines the number of Memcached instances
    // +operator-sdk:csv:customresourcedefinitions:type=spec
    Size int32 `json:"size,omitempty"`

    // +operator-sdk:csv:customresourcedefinitions:type=spec
    ContainerPort int32 `json:"containerPort,omitempty"`
}
```

```yaml
type: object
x-kubernetes-validations:
- message: Value is immutable
  rule: self == oldSelf
```

Validating Admission Webhooks

- “An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.”
- Validating Admission Webhook contains logic to Deny or Admit the requests to the Kubernetes API
- We can treat Validating Webhooks as a simple stateless web-server or fully capable controller
- Should be used only for really complex validation logic that can’t be covered by CEL validation rules
- Complex to maintain (deployment, certs, build process)
- Validation webhooks is executed just after mutating webhook
- **[Kubebuilder]** As for now it’s not possible to write mutating and validating admission webhooks for non custom resources

Validating Admission Webhooks



https://kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/

Defaulting

“We never want to change or override a value that was provided by the user, if they requested something invalid, they should get an error: validation!”

Defaulting - Defaulter function

```go
var _ webhook.Defaulter = &CronJob{}

// Default implements webhook.Defaulter so a webhook will be registered for the type
func (r *CronJob) Default() {
    cronjoblog.Info("default", "name", r.Name)

    if r.Spec.ConcurrencyPolicy == "" {
        r.Spec.ConcurrencyPolicy = AllowConcurrent
    }
    if r.Spec.Suspend == nil {
        r.Spec.Suspend = new(bool)
    }
    if r.Spec.SuccessfulJobsHistoryLimit == nil {
        r.Spec.SuccessfulJobsHistoryLimit = new(int32)
        *r.Spec.SuccessfulJobsHistoryLimit = 3
    }
    if r.Spec.FailedJobsHistoryLimit == nil {
        r.Spec.FailedJobsHistoryLimit = new(int32)
        *r.Spec.FailedJobsHistoryLimit = 1
    }
}
```

https://book.kubebuilder.io/cronjob-tutorial/webhook-implementation

Defaulting - OpenAPI v3 schema

- Defaulting is executed at APIServer level, just after reading the data from ETCD
- Defaulting happens on the object
  - in the request to the API server using the request version defaults,
  - when reading from etcd using the storage version defaults,
  - after mutating admission plugins with non-empty patches using the admission webhook object version defaults.
- “Defaults applied when reading data from etcd are not automatically written back to etcd. An update request via the API is required to persist those defaults back into etcd”

```yaml
schema:
  openAPIV3Schema:
    type: object
    properties:
      spec:
        type: object
        properties:
          cronSpec:
            type: string
            pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
            default: "5 0 * * *"
          image:
            type: string
          replicas:
            type: integer
            minimum: 1
            maximum: 10
            default: 1
```

https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/#defaulting

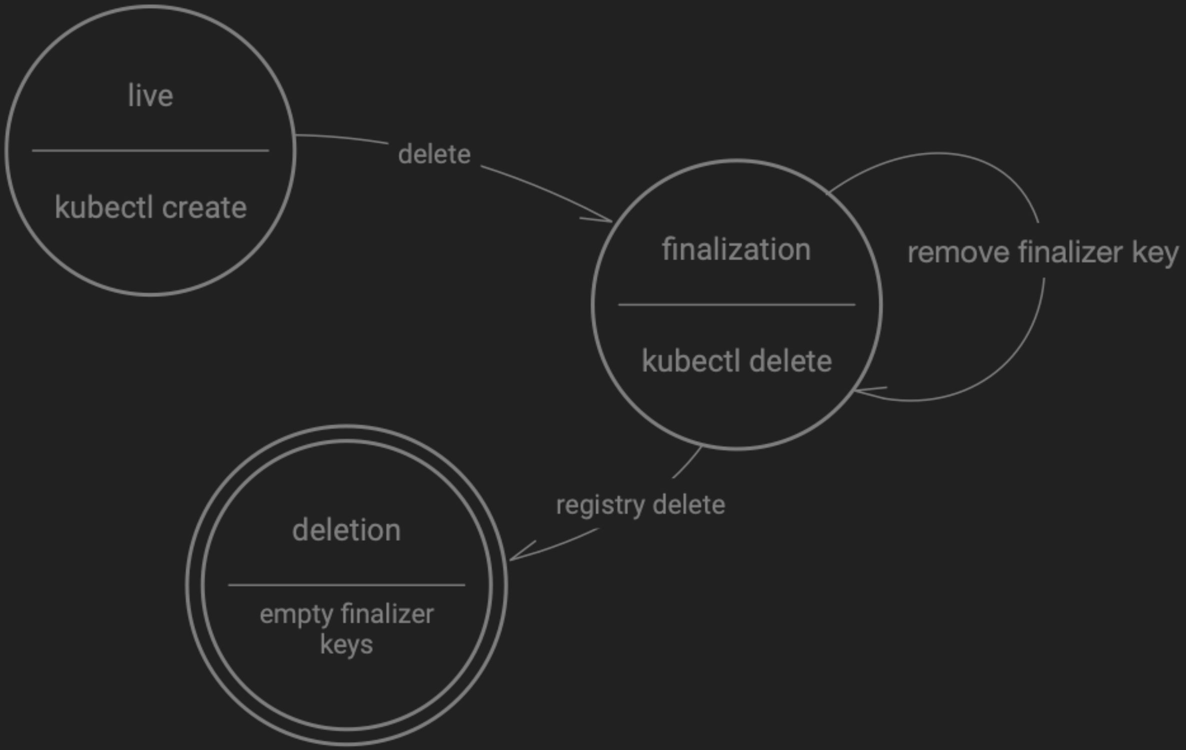
Finalizers

Finalizers

- “Finalizers are namespaced keys that tell Kubernetes to wait until specific conditions are met before it fully deletes resources marked for deletion”
- “Finalizers alert controllers to clean up resources the deleted object owned”
- Finalizers are specified in *.metadata.finalizer* block
- “When you attempt to delete the resource, the API server handling the delete request notices the values in the finalizers field and does the following:”
  a. Modifies the object to add a *metadata.deletionTimestamp* field with the time you started the deletion.
  b. *P*revents the object from being removed until all items are removed from its metadata.finalizers field
  c. Returns a 202 status code *(HTTP "Accepted")*

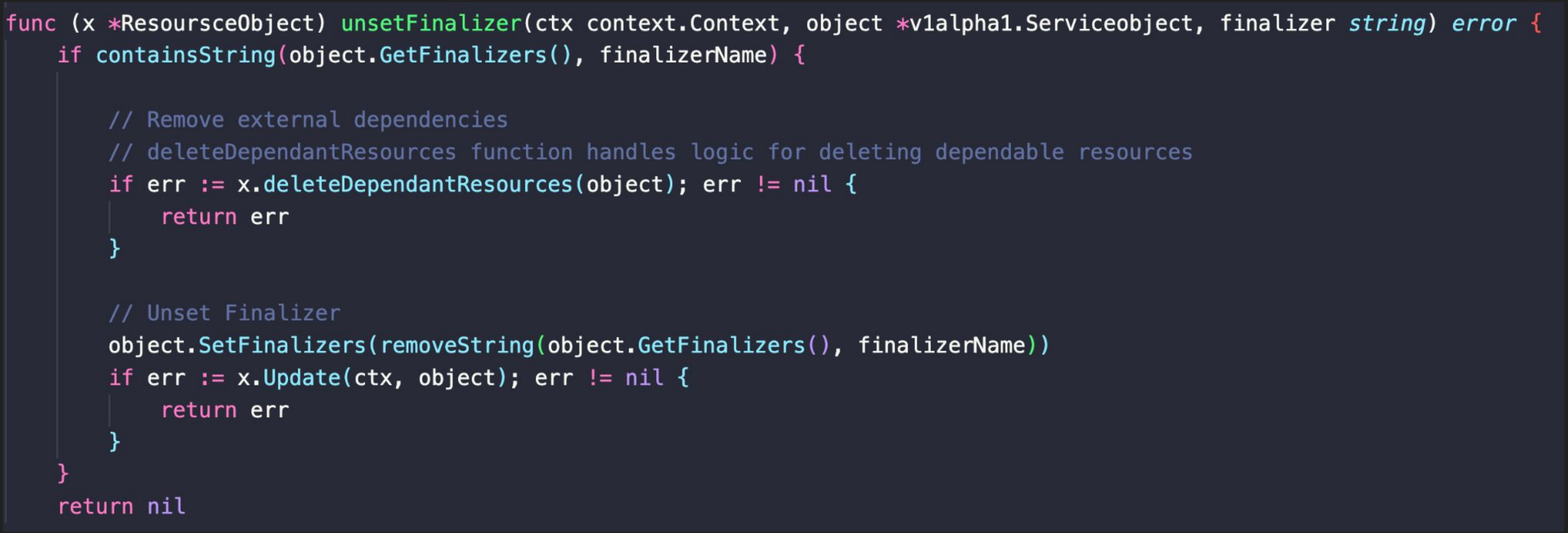
https://kubernetes.io/docs/concepts/overview/working-with-objects/finalizers/#owners-labels-finalizers

Finalizers - State Diagram



https://kubernetes.io/blog/2021/05/14/using-finalizers-to-control-deletion/

Finalizers - implementation (trimmed)



```go
func (x *ResoursceObject) unsetFinalizer(ctx context.Context, object *v1alpha1.Serviceobject, finalizer string) error {
    if containsString(object.GetFinalizers(), finalizerName) {

        // Remove external dependencies
        // deleteDependantResources function handles logic for deleting dependable resources
        if err := x.deleteDependantResources(object); err != nil {
            return err
        }

        // Unset Finalizer
        object.SetFinalizers(removeString(object.GetFinalizers(), finalizerName))
        if err := x.Update(ctx, object); err != nil {
            return err
        }
    }
    return nil
```

Test Clusters

Test Clusters & Testing Components

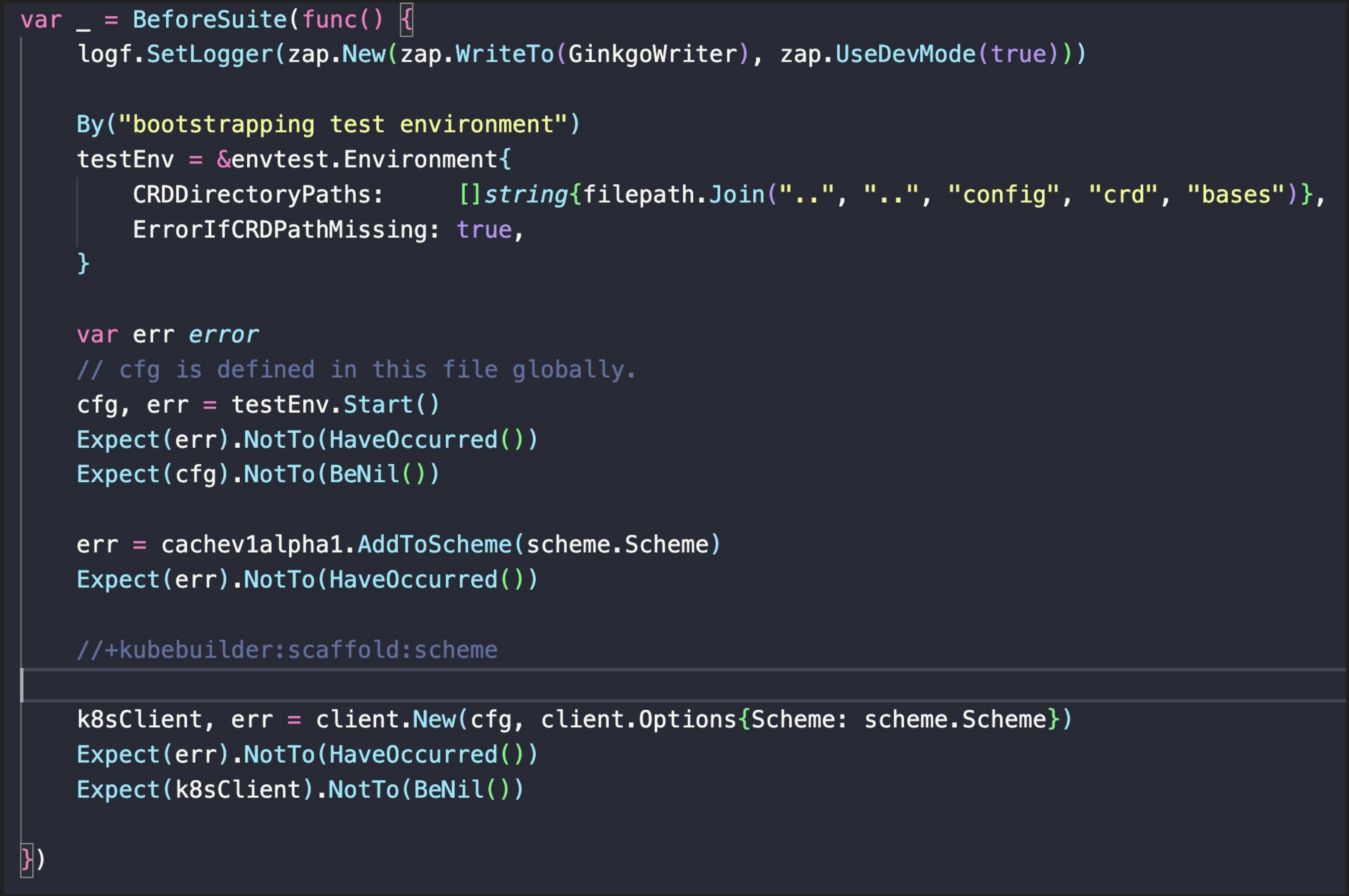
kind

Ginkgo

Gomega

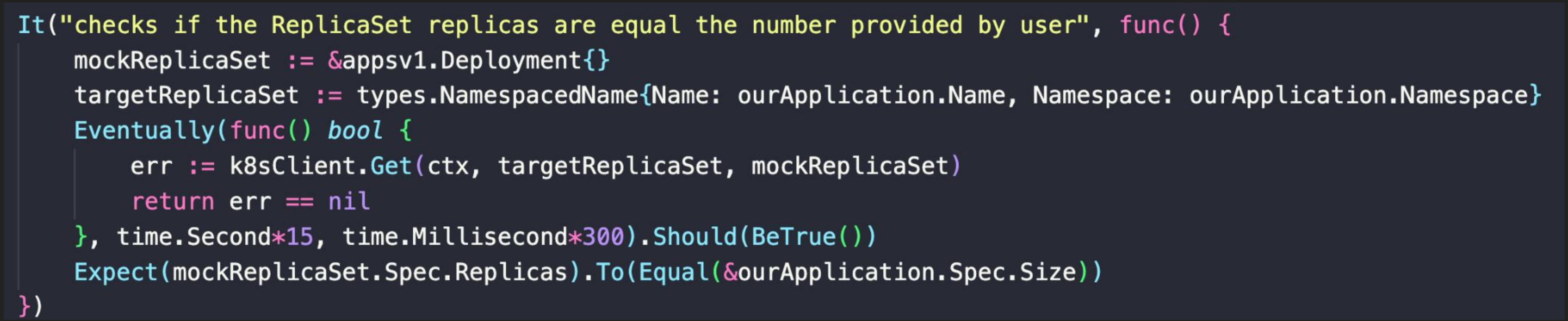
Test Clusters & Testing Components

- Prepare at least two different Kind setups
  a. Standard setup: 1 master, 3 nodes
  b. Large setup: 3/5 masters, 10 nodes
- Ginkgo + Gomega + EnvTest makes a perfect mix for integration tests
- At some point of time setup load tests/stress tests as a part of testing routine. Kubernetes controllers behave differently with the large amount of objects or requests
- Use ginkgo --until-it-fails to identify tests that are flaky
- Check *Kubernetes-sig* repositories for real world examples of integration testing with Kind
- Maintain E2E cluster which is the clone of the production one
- Fuzz fields that can benefit from it

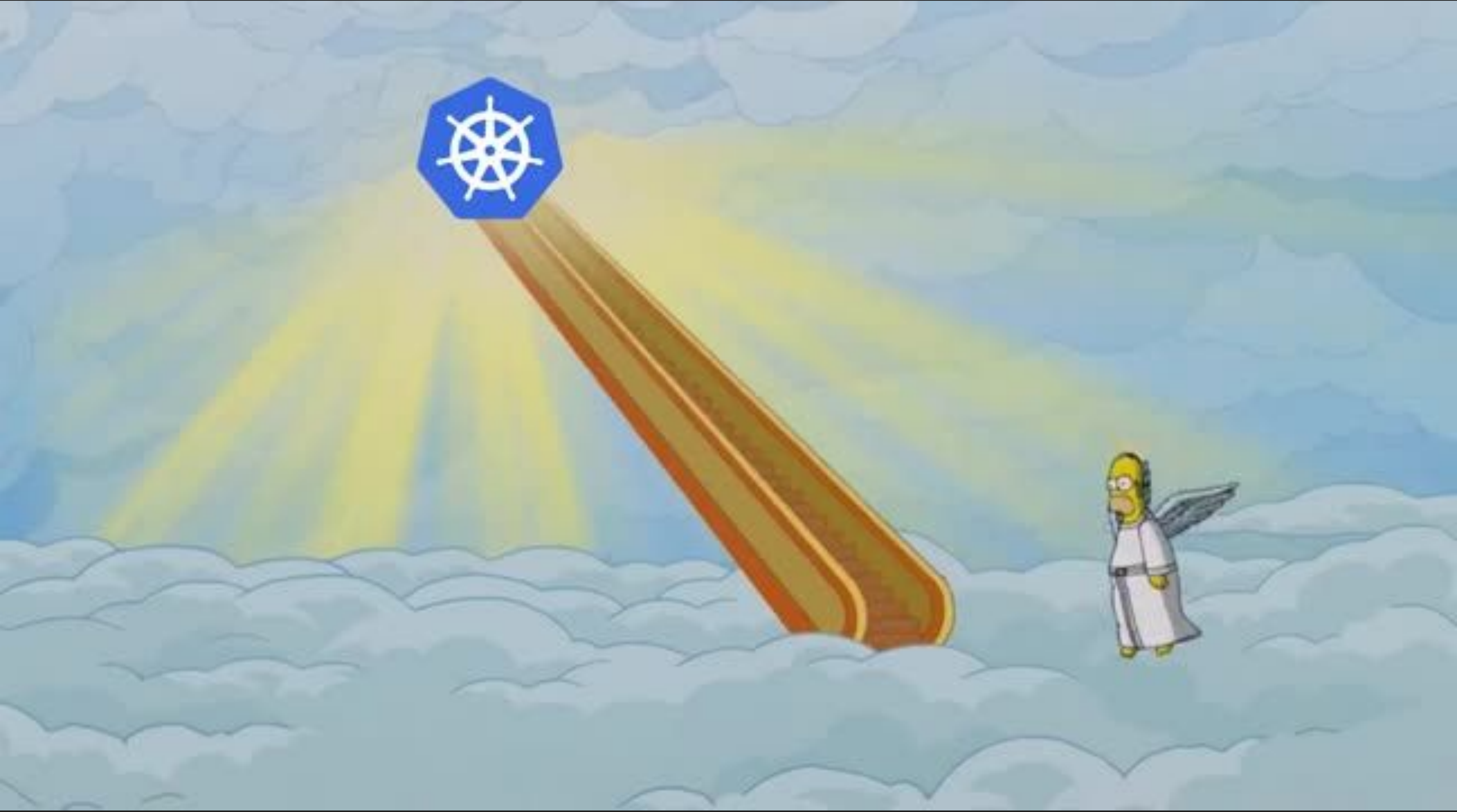


```go
var _ = BeforeSuite(func() {
    logf.SetLogger(zap.New(zap.WriteTo(GinkgoWriter), zap.UseDevMode(true)))

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

    var err error
    // cfg is defined in this file globally.
    cfg, err = testEnv.Start()
    Expect(err).NotTo(HaveOccurred())
    Expect(cfg).NotTo(BeNil())

    err = cachev1alpha1.AddToScheme(scheme.Scheme)
    Expect(err).NotTo(HaveOccurred())

    //+kubebuilder:scaffold:scheme

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

})
```

Ginkgo example



```go
It("checks if the ReplicaSet replicas are equal the number provided by user", func() {
    mockReplicaSet := &appsv1.Deployment{}
    targetReplicaSet := types.NamespacedName{Name: ourApplication.Name, Namespace: ourApplication.Namespace}
    Eventually(func() bool {
        err := k8sClient.Get(ctx, targetReplicaSet, mockReplicaSet)
        return err == nil
    }, time.Second*15, time.Millisecond*300).Should(BeTrue())
    Expect(mockReplicaSet.Spec.Replicas).To(Equal(&ourApplication.Spec.Size))
})
```



Useful links

- Difference between controller and operator - https://github.com/kubeflow/training-operator/issues/300
- Explanation of Kubernetes validation against objects/schemas - https://danielmangum.com/posts/how-kubernetes-validates-custom-resources/
- How finalizers work + simple implementation - https://gogolok.github.io/posts/kubernetes-finalizers-in-custom-resources/
- Implementation of simple Kubernetes webhook - https://slack.engineering/simple-kubernetes-webhook/
- How to develop a Robust Operator for Day-2 (Lesson Learned on KubeVirt/HCO) - https://www.youtube.com/watch?v=vbDX4gOQb5E

Thank you!

https://www.linkedin.com/in/patryk-wasielewski/