KubeCon CloudNativeCon
North America 2021

RESILIENCE
REALIZED

Customizing Kustomize with Client-Side Custom Resources

Katrina Verey (@KnVerey), Apple
Jeff Regan (@monopole)

Overview

🤔 What is Kustomize?

Fundamentals for extension developers

🧩 Client-side custom resources

Kustomize extension use cases, benefits and caveats

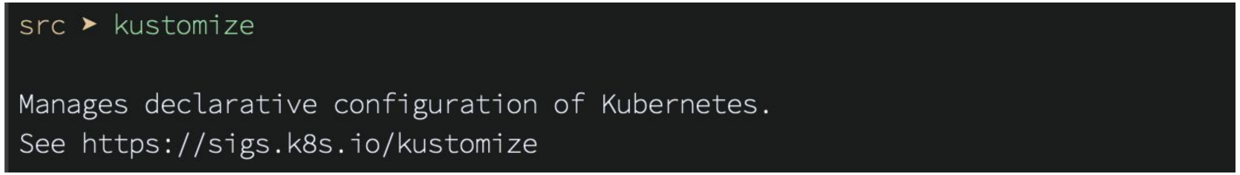
🔨 Build your own extension

Examples, tools and best practices

What is Kustomize?

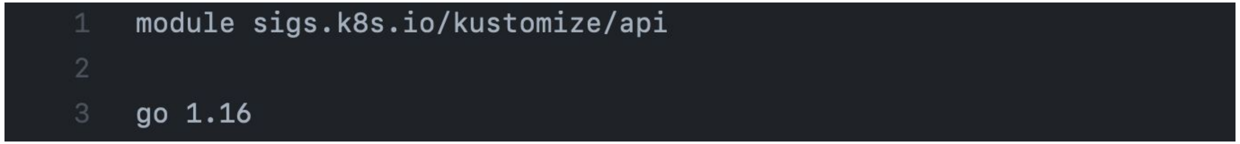
What is Kustomize?

A free-standing utility



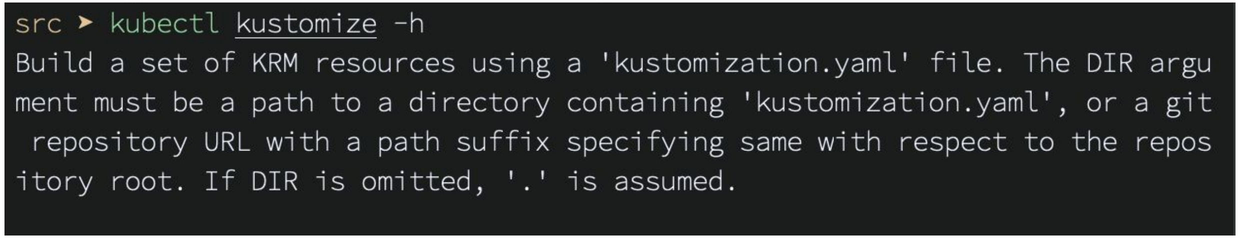
```
src ➤ kustomize

Manages declarative configuration of Kubernetes.
See https://sigs.k8s.io/kustomize
```

A Go module anyone can use



```
module sigs.k8s.io/kustomize/api

go 1.16
```

A kubectl subcommmand



```
src ➤ kubectl kustomize -h
Build a set of KRM resources using a 'kustomization.yaml' file. The DIR argu
ment must be a path to a directory containing 'kustomization.yaml', or a git
 repository URL with a path suffix specifying same with respect to the repos
itory root. If DIR is omitted, '.' is assumed.
```

What is Kustomize?

A **configuration stream editor** that

1. works with k8s-style **resource objects**
2. supports **variants** (e.g. prod, staging, dev)
3. leverages **git** concepts
4. is **extensible**

What is Kustomize?

KubeCon CloudNativeCon North America 2021

Guiding principles

- Configuration at rest directly usable by k8s

  No templates. No domain specific language. Config is raw data.

- *Edits* to configuration expressed as *k8s objects*

  Kustomize gets its instructions from k8s objects.

What is Kustomize?

It’s all Kubernetes YAML

```
$ tree helloWorld
helloWorld
├── configMap.yaml
├── deployment.yaml
├── kustomization.yaml   ← Drop this in to
└── service.yaml              describes edits
```

What is Kustomize?

It’s all Kubernetes YAML

service.yaml

```
kind: Service
metadata:
 name: wordpress
spec:
 ports:
    - port: 389
    selector:
      app: wordpress
```

kustomization.yaml

```
kind: Kustomization
resources:
- service.yaml

namePrefix: my-

commonLabels:
  app: demo
```

kustomize build

/dev/stdout

```
kind: Service
metadata:
 name: my-wordpress
 labels:
  app: demo
spec:
 ports:
    - port: 389
    selector:
      app: demo
```

What is Kustomize?

It’s all Kubernetes YAML

```
$ kustomize build helloWorld | \
      kubectl apply -f -
```

What is Kustomize?

It’s all Kubernetes YAML

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- deployment.yaml

namePrefix: bob-
```

this field
is internally expanded
to **this transformer config**

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- deployment.yaml

transformers:
- |-
  apiVersion: builtin
  kind: PrefixSuffixTransformer
  metadata:
    name: myFancyNamePrefixer
  prefix: bob-
  fieldSpecs:
  - path: metadata/name
```

What is Kustomize?

Edits are expressed as client-side custom resources

Built-in transformer

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

transformers:
- |-
  apiVersion: builtin
  kind: PrefixSuffixTransformer
  metadata:
    name: myFancyNamePrefixer
  prefix: bob-
  fieldSpecs:
  - path: metadata/name
```

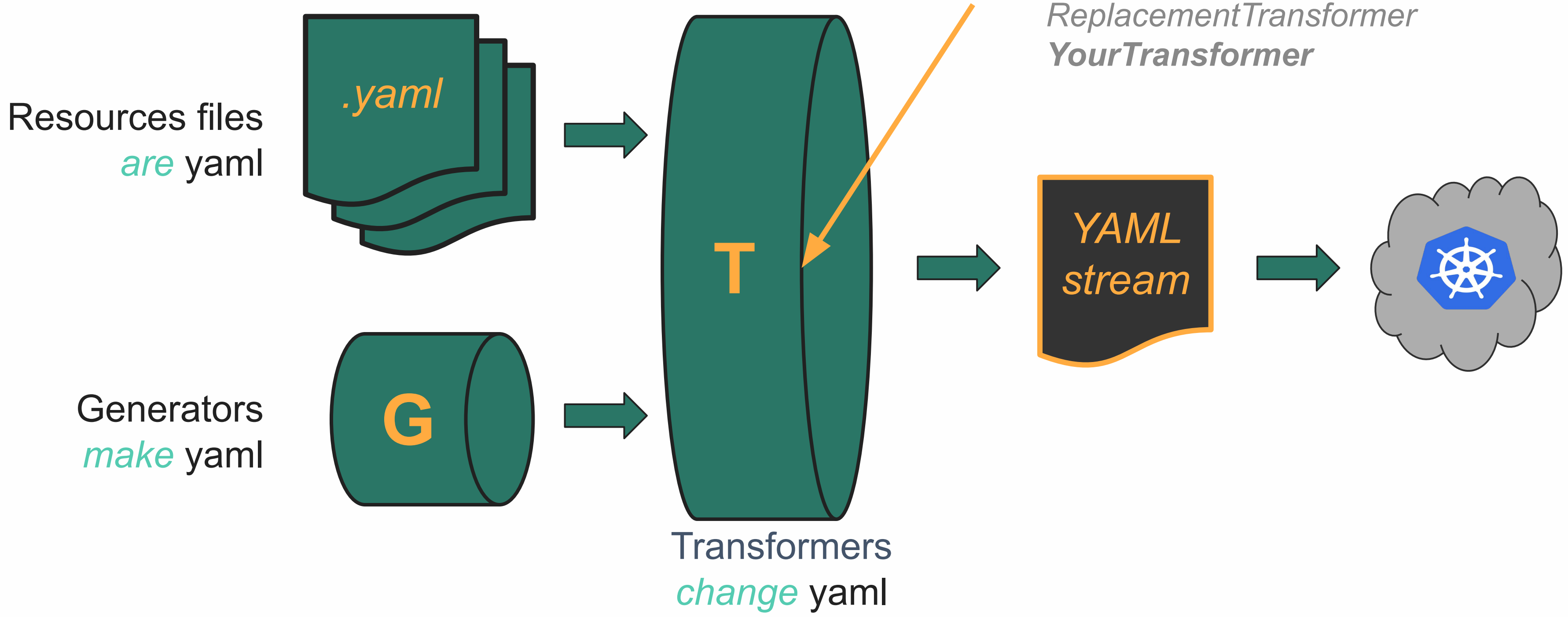
External generator

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- |-
  apiVersion: example.com/v1alpha1
  kind: JavaSpringBoot
  metadata:
    name: my-app
  spec:
    image: apps.myco.com/javaspringboot/app:1.0
    domain: my-app.myco.com
    ...
```

What is Kustomize?

It's all Kubernetes YAML

e.g. *PatchTransformer*
NamespaceTransformer
PrefixSuffixTransformer
ReplacementTransformer
YourTransformer



Client-side custom resources

Client-side custom resources

What can they do?

- **Generate**
- Transform
- Validate

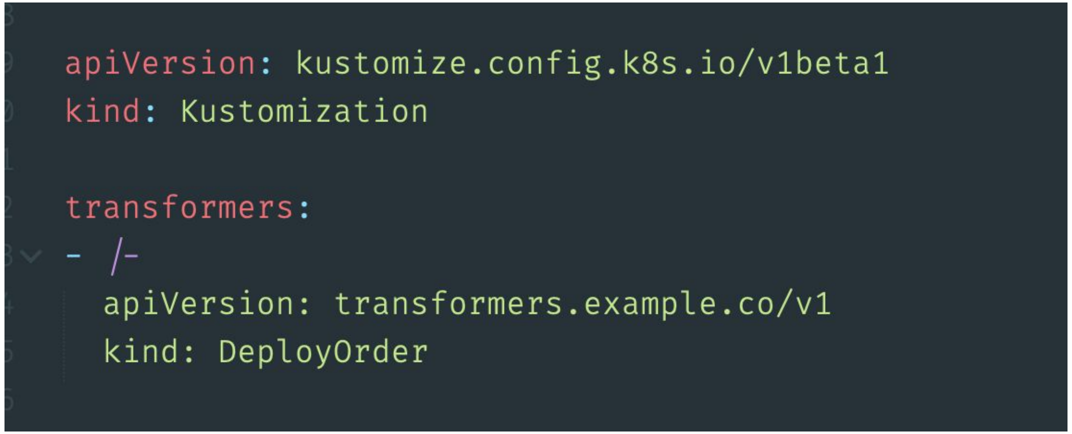
```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- |-
  apiVersion: example.com/v1alpha1
  kind: JavaSpringBoot
  metadata:
    name: my-app
  spec:
    image: apps.myco.com/javaspringboot/app:1.0
    domain: my-app.myco.com
     ...
```

Deployment
Ingress
Service
NetworkPolicy

Client-side custom resources

What can they do?

- Generate
- **Transform**
- Validate



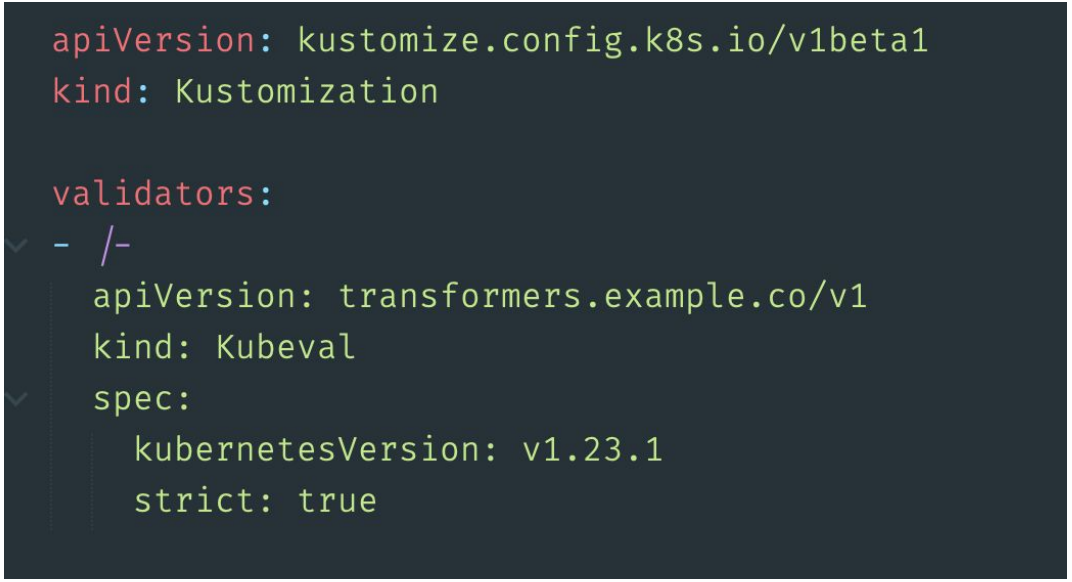
```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

transformers:
- |-
  apiVersion: transformers.example.co/v1
  kind: DeployOrder
```

Client-side custom resources

KubeCon CloudNativeCon North America 2021

What can they do?

- Generate
- Transform
- **Validate**



```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

validators:
- |-
  apiVersion: transformers.example.co/v1
  kind: Kubeval
  spec:
    kubernetesVersion: v1.23.1
    strict: true
```

Client-side custom resources

```
kind: LogExporter
```

```
kind: GCPIngress
```

```
kind: JavaSpringBoot
```

```
kind: HTTPLoadBalancer
```

```
kind: DomainInjector
```

```
kind: DeployOrder
```

```
kind: Helm
```

```
kind: MyKMS
```

```
kind: Kubeval
```

```
kind: StagingTransformer
```

```
kind: MyCoolSidecar
```

Client-side custom resources

Benefits

- Familiar, declarative KRM APIs
- No templating
- No new language
- Open standard

Client-side custom resources

Benefits

- Versus server-side abstractions:
  - Nothing to install in-cluster
  - End users retain control
  - It’s just YAML in source control
  - Faster development cycle

Client-side custom resources

Considerations

- Should not have side-effects in-cluster (or elsewhere)
- Cannot prevent users from altering the output
- Not suitable for policy enforcement
- Risk added to `kustomize build`

Build your own extension

Build your own extension

History of Kustomize extension alphas

- Legacy plugins
  - ~~Go plugins~~
  - ~~Exec plugins~~

- KRM Functions Specification
  - Exec functions
  - ~~Starlark functions~~
  - **Container functions**

Build your own extension

KubeCon | CloudNativeCon
North America 2021

Future

- Kustomize Plugin Graduation: k/enhancements#2953
- Kustomize Plugin Catalog: k/enhancements#2906
- Kustomize Plugin Composition: k/enhancements#2299

Build your own extension

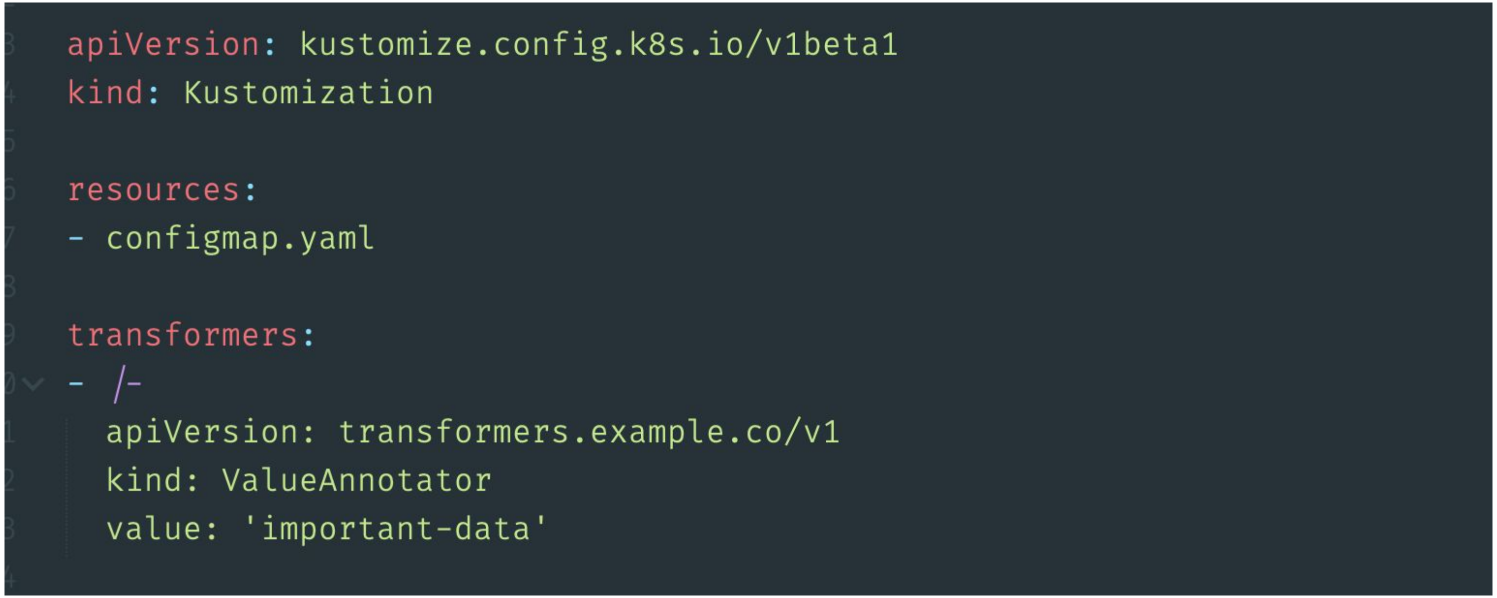
Recommended approach today

- Container-based “KRM functions”
- Developed with function framework package

  (in kyaml/fn/framework)

Build your own extension

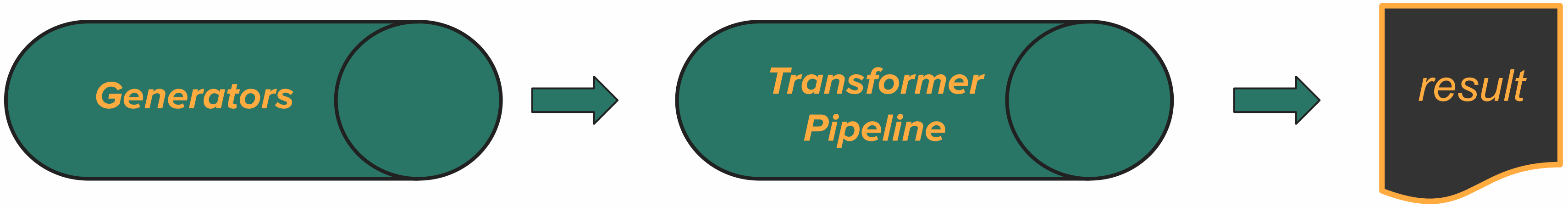
Basic example: End-user view



```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- configmap.yaml

transformers:
- |-
  apiVersion: transformers.example.co/v1
  kind: ValueAnnotator
  value: 'important-data'
```

Build your own extension

A KRM-driven pipeline



Resources field

+

e.g. ConfigMapGenerator

+

Other generator

PatchTransformer

↓

NamespaceTransformer

e.g. ↓

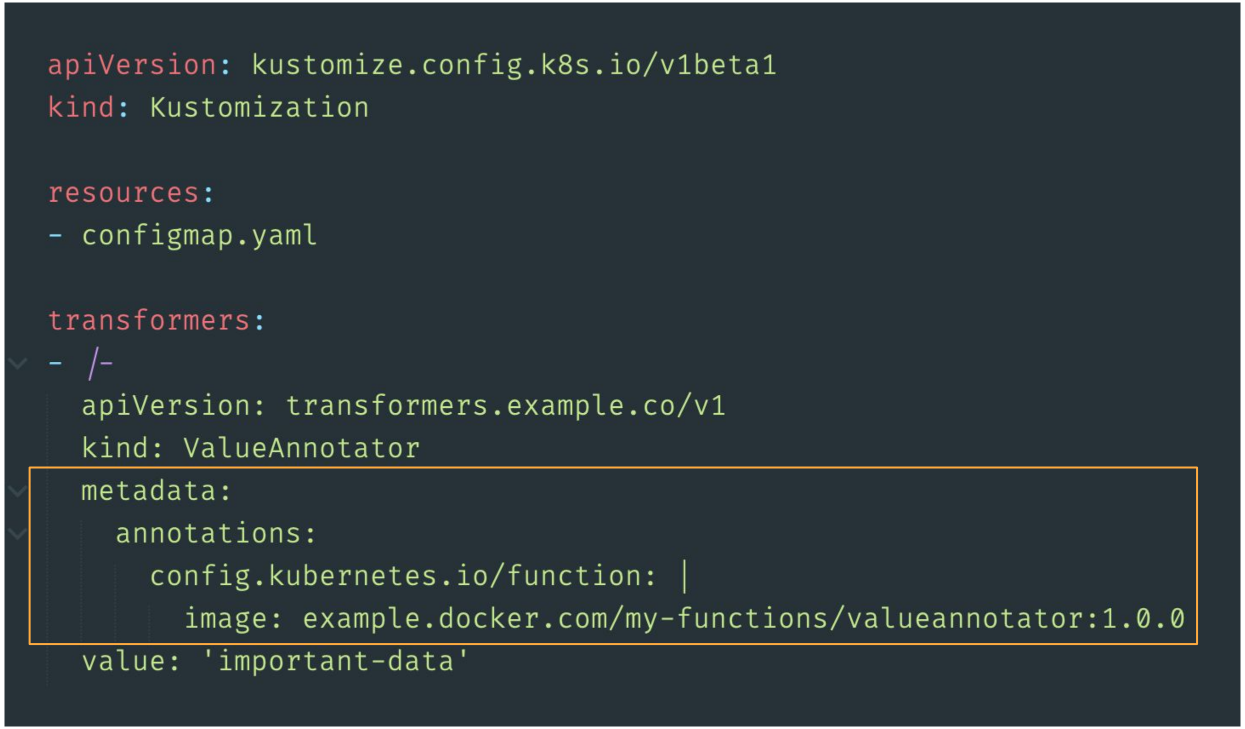
PrefixSuffixTransformer

↓

Other transformer

Build your own extension

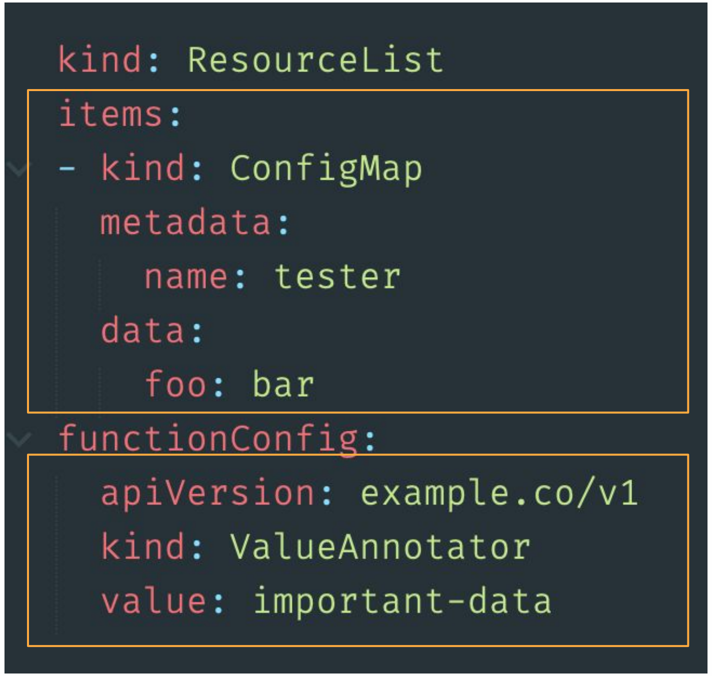
Basic example: End-user view



```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- configmap.yaml

transformers:
- |-
  apiVersion: transformers.example.co/v1
  kind: ValueAnnotator
  metadata:
    annotations:
      config.kubernetes.io/function: |
        image: example.docker.com/my-functions/valueannotator:1.0.0
  value: 'important-data'
```

Build your own extension

Basic example: Input



```yaml
kind: ResourceList
items:
- kind: ConfigMap
  metadata:
    name: tester
  data:
    foo: bar
functionConfig:
  apiVersion: example.co/v1
  kind: ValueAnnotator
  value: important-data
```

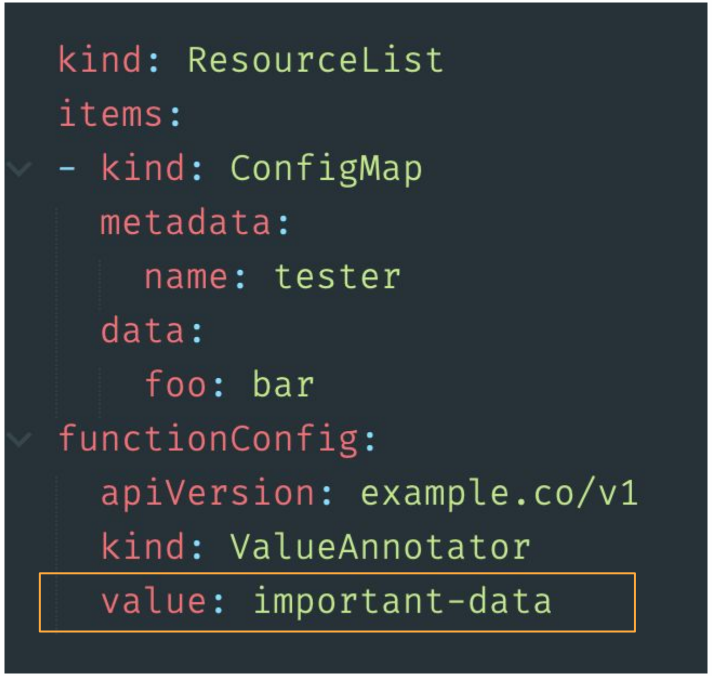
← *Resources to process*

← *Our plugin config CR*

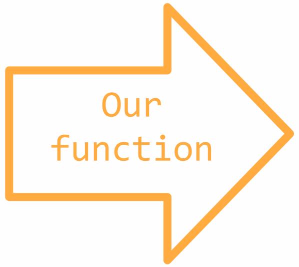
Build your own extension

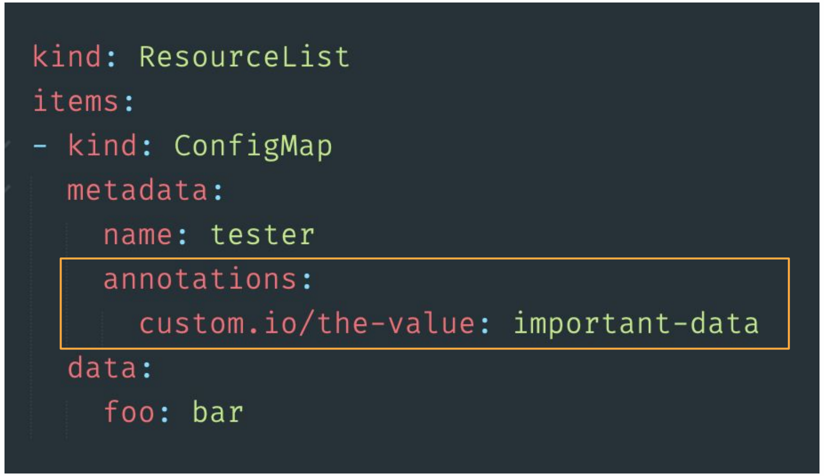
KubeCon | CloudNativeCon
North America 2021

Basic example: desired behavior



```yaml
kind: ResourceList
items:
- kind: ConfigMap
  metadata:
    name: tester
  data:
    foo: bar
functionConfig:
  apiVersion: example.co/v1
  kind: ValueAnnotator
  value: important-data
```



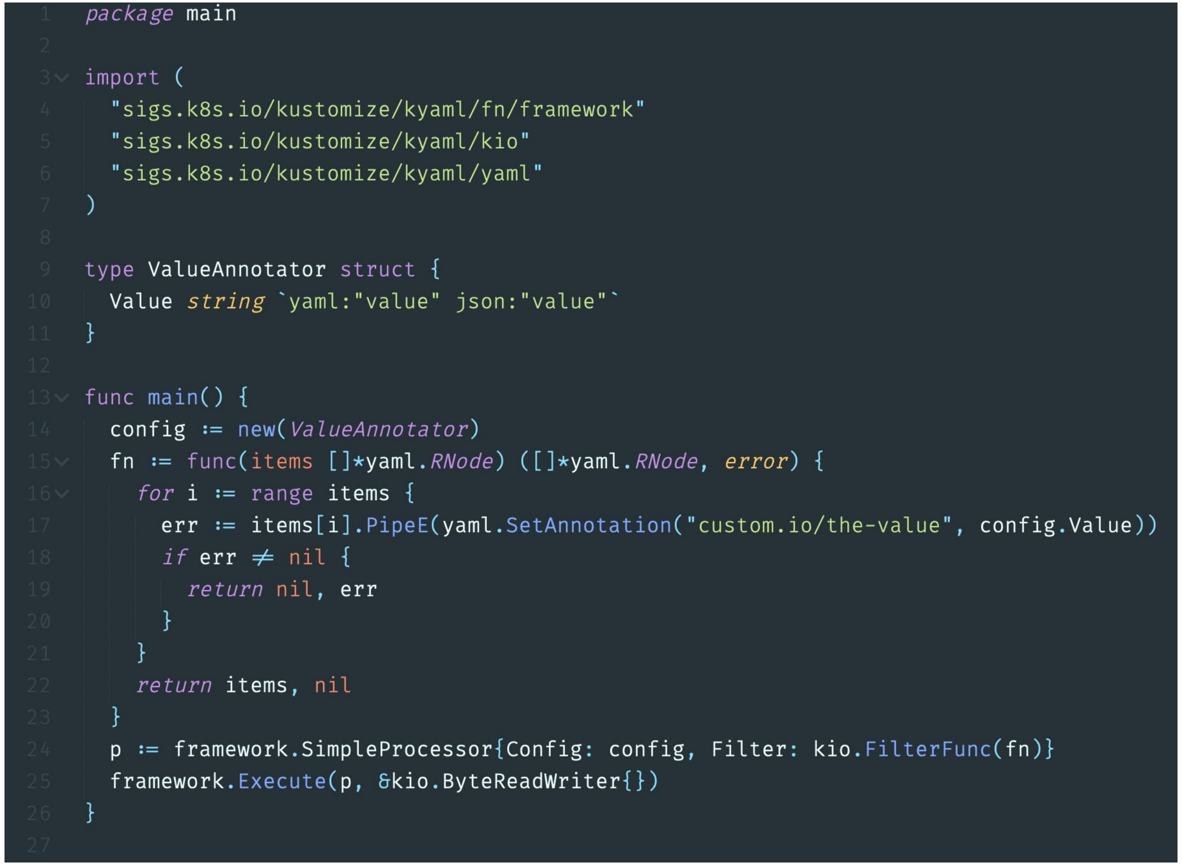


```yaml
kind: ResourceList
items:
- kind: ConfigMap
  metadata:
    name: tester
    annotations:
      custom.io/the-value: important-data
  data:
    foo: bar
```

Build your own extension

KubeCon CloudNativeCon North America 2021

Basic example

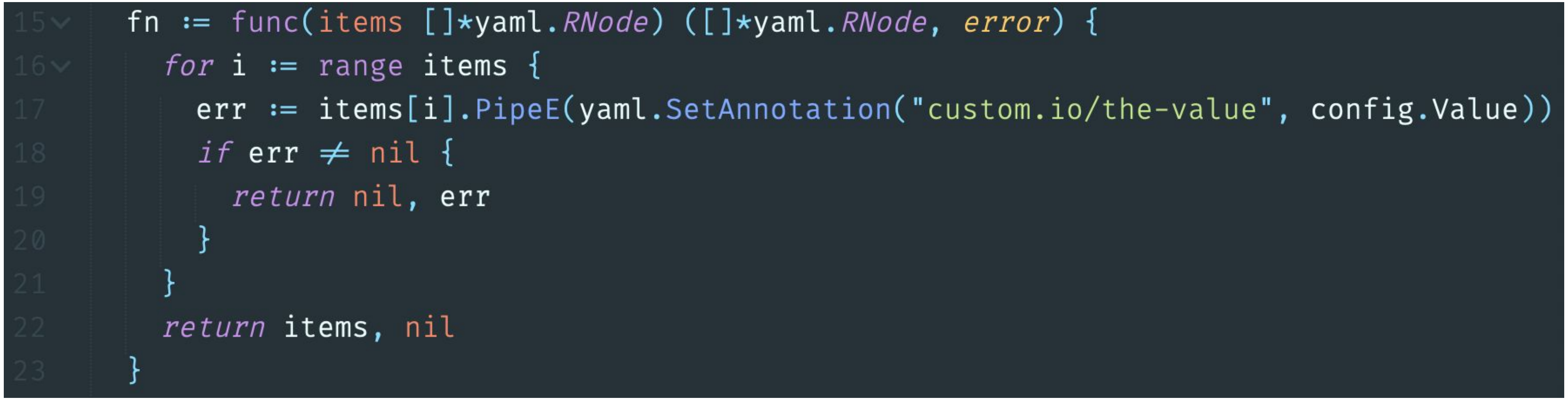


```go
package main

import (
  "sigs.k8s.io/kustomize/kyaml/fn/framework"
  "sigs.k8s.io/kustomize/kyaml/kio"
  "sigs.k8s.io/kustomize/kyaml/yaml"
)

type ValueAnnotator struct {
  Value string `yaml:"value" json:"value"`
}

func main() {
  config := new(ValueAnnotator)
  fn := func(items []*yaml.RNode) ([]*yaml.RNode, error) {
    for i := range items {
      err := items[i].PipeE(yaml.SetAnnotation("custom.io/the-value", config.Value))
      if err != nil {
        return nil, err
      }
    }
    return items, nil
  }
  p := framework.SimpleProcessor{Config: config, Filter: kio.FilterFunc(fn)}
  framework.Execute(p, &kio.ByteReadWriter{})
}
```

Build your own extension

Basic example

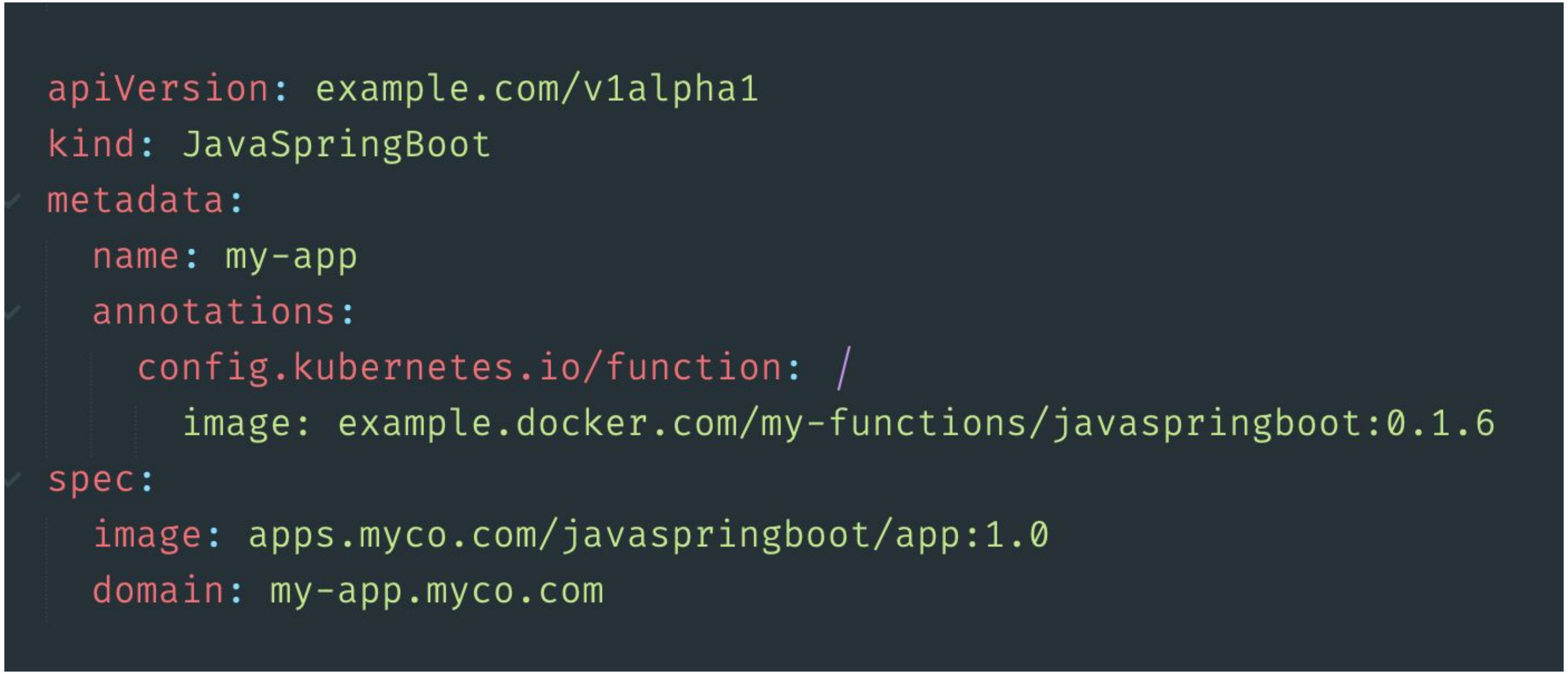
Core logic



```
fn := func(items []*yaml.RNode) ([]*yaml.RNode, error) {
  for i := range items {
    err := items[i].PipeE(yaml.SetAnnotation("custom.io/the-value", config.Value))
    if err ≠ nil {
      return nil, err
    }
  }
  return items, nil
}
```

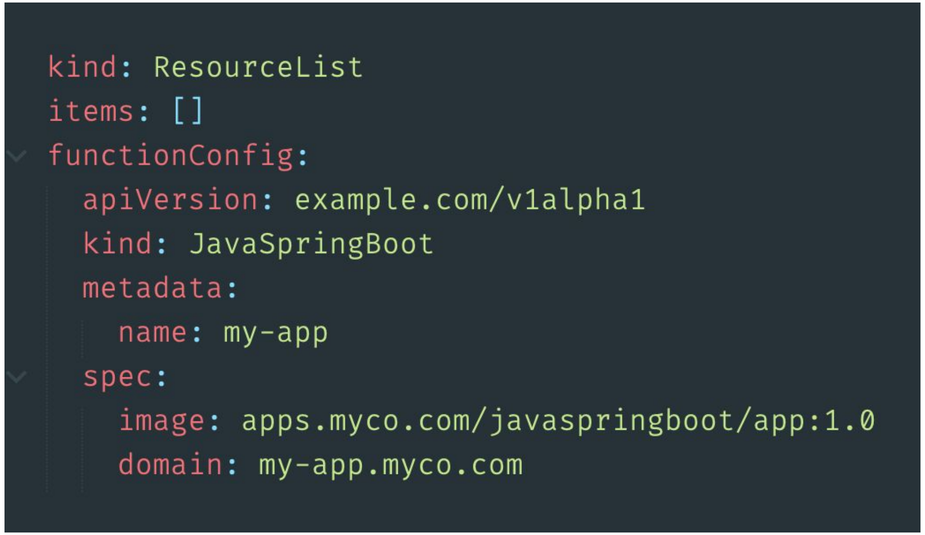
Build your own extension

Advanced example

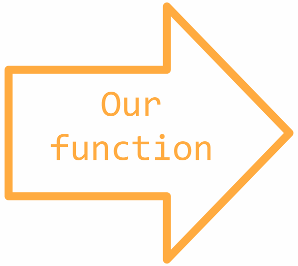


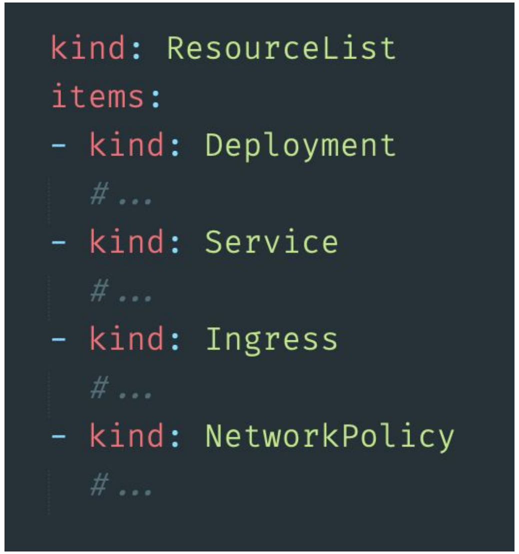
```
apiVersion: example.com/v1alpha1
kind: JavaSpringBoot
metadata:
  name: my-app
  annotations:
    config.kubernetes.io/function: |
      image: example.docker.com/my-functions/javaspringboot:0.1.6
spec:
  image: apps.myco.com/javaspringboot/app:1.0
  domain: my-app.myco.com
```

Advanced example: desired behavior



```yaml
kind: ResourceList
items: []
functionConfig:
  apiVersion: example.com/v1alpha1
  kind: JavaSpringBoot
  metadata:
    name: my-app
  spec:
    image: apps.myco.com/javaspringboot/app:1.0
    domain: my-app.myco.com
```





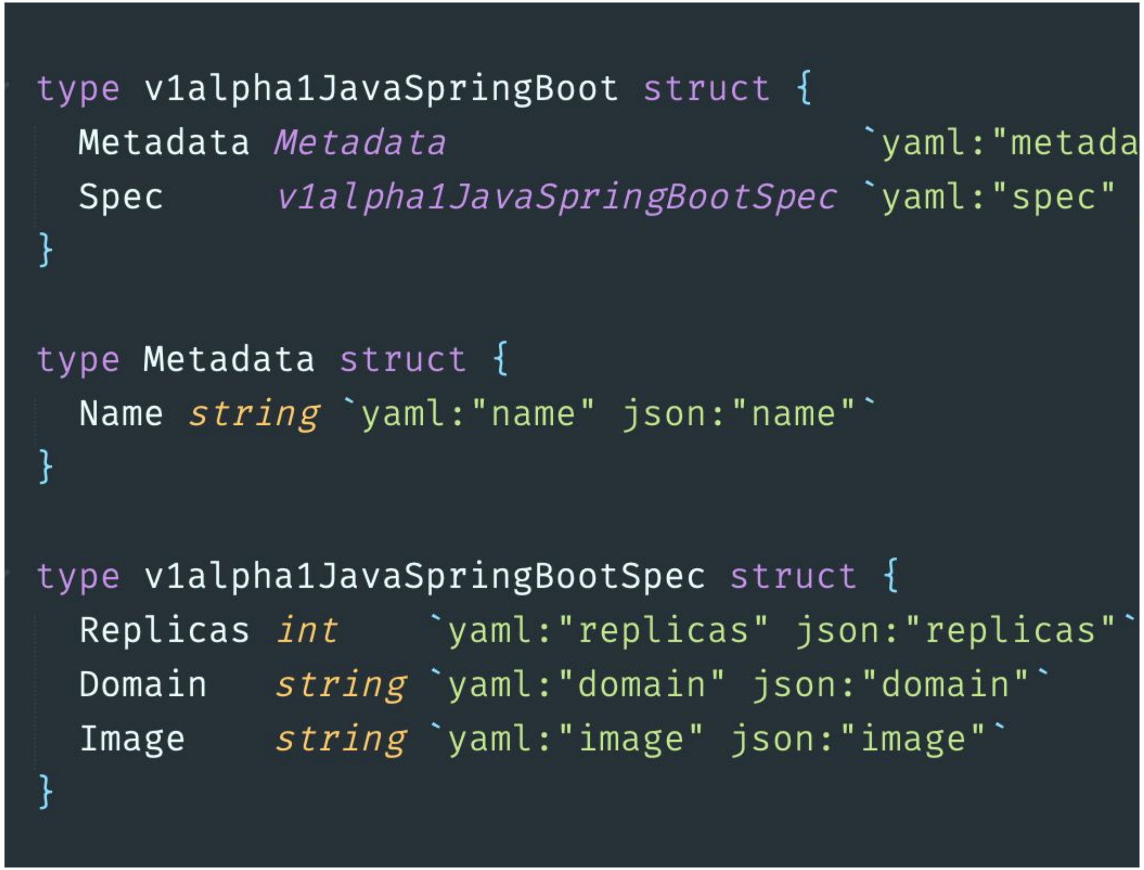
```yaml
kind: ResourceList
items:
- kind: Deployment
  # ...
- kind: Service
  # ...
- kind: Ingress
  # ...
- kind: NetworkPolicy
  # ...
```

Build your own extension

KubeCon CloudNativeCon North America 2021

Advanced example

Define the type for your API



```
type v1alpha1JavaSpringBoot struct {
  Metadata Metadata                    `yaml:"metada
  Spec     v1alpha1JavaSpringBootSpec `yaml:"spec"
}

type Metadata struct {
  Name string `yaml:"name" json:"name"`
}

type v1alpha1JavaSpringBootSpec struct {
  Replicas int      `yaml:"replicas" json:"replicas"`
  Domain   string `yaml:"domain" json:"domain"`
  Image    string `yaml:"image" json:"image"`
}
```

Build your own extension

Advanced example

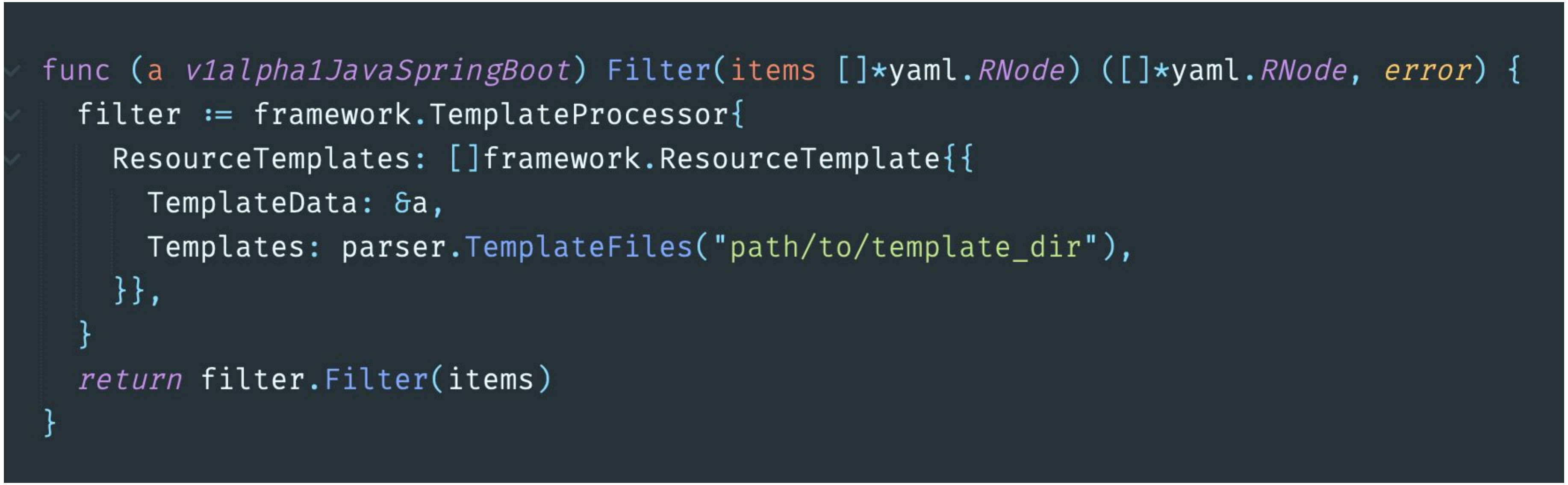
Implement `Filter` on your type

```go
func (a v1alpha1JavaSpringBoot) Filter(items []*yaml.RNode) ([]*yaml.RNode, error) {
	// your logic here
}
```

Build your own extension

Advanced example

`framework.TemplateProcessor` can help!

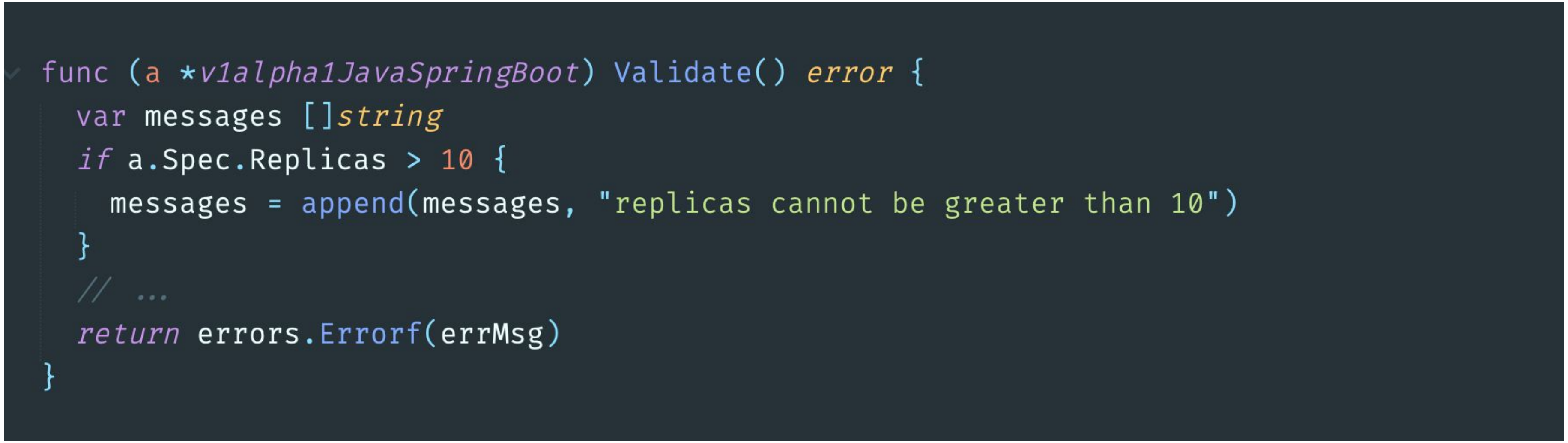


```go
func (a v1alpha1JavaSpringBoot) Filter(items []*yaml.RNode) ([]*yaml.RNode, error) {
  filter := framework.TemplateProcessor{
    ResourceTemplates: []framework.ResourceTemplate{{
      TemplateData: &a,
      Templates: parser.TemplateFiles("path/to/template_dir"),
    }},
  }
  return filter.Filter(items)
}
```

Build your own extension

Advanced example

It’s easy to… add **validation**

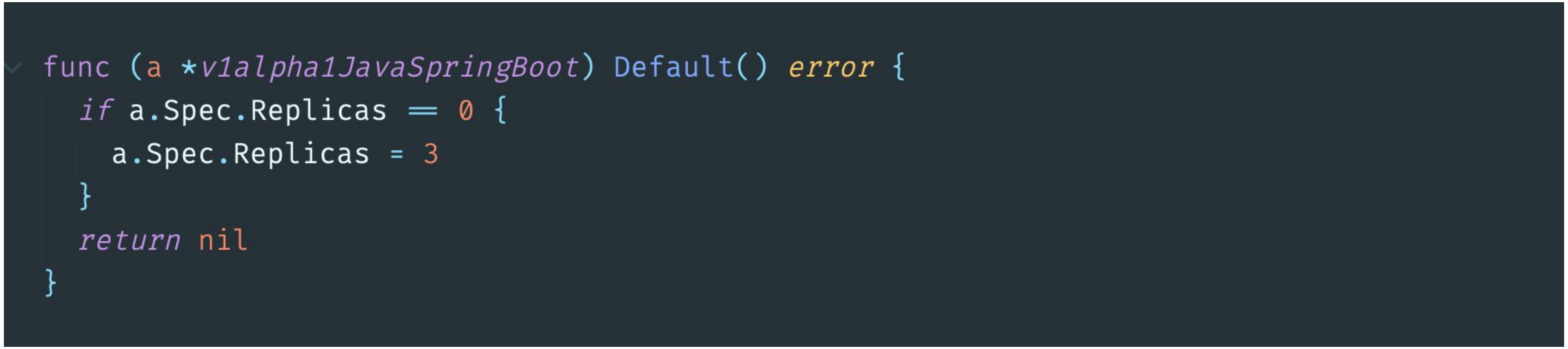


```go
func (a *v1alpha1JavaSpringBoot) Validate() error {
  var messages []string
  if a.Spec.Replicas > 10 {
    messages = append(messages, "replicas cannot be greater than 10")
  }
  // ...
  return errors.Errorf(errMsg)
}
```

Build your own extension

Advanced example

It’s easy to… add **defaulting**

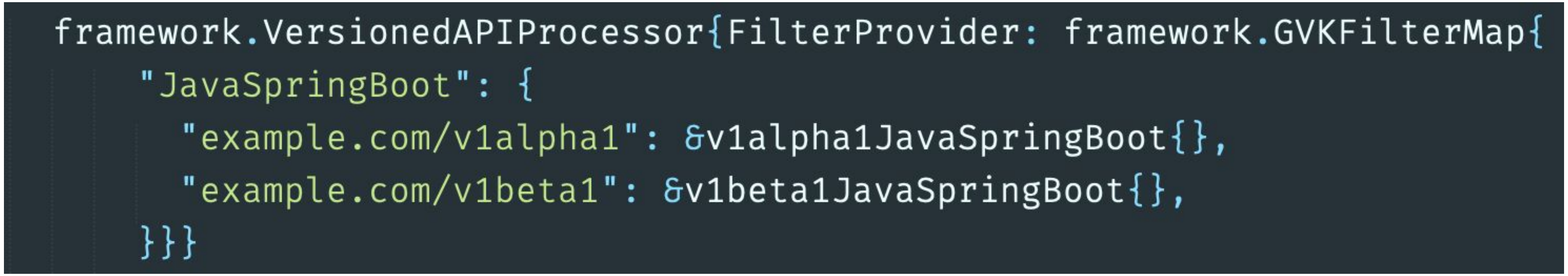


```go
func (a *v1alpha1JavaSpringBoot) Default() error {
  if a.Spec.Replicas == 0 {
    a.Spec.Replicas = 3
  }
  return nil
}
```

Build your own extension

Advanced example

It's easy to… add **multi-version support**



```
framework.VersionedAPIProcessor{FilterProvider: framework.GVKFilterMap{
    "JavaSpringBoot": {
      "example.com/v1alpha1": &v1alpha1JavaSpringBoot{},
      "example.com/v1beta1": &v1beta1JavaSpringBoot{},
    }}}
```

Build your own extension

Additional tools

- Support for patching **CRDs**
- Ability to target **patch** templates to **containers**
- Suite of **selectors** and **matchers**
- **Dockerfile** generation
- Much more!

Best practices

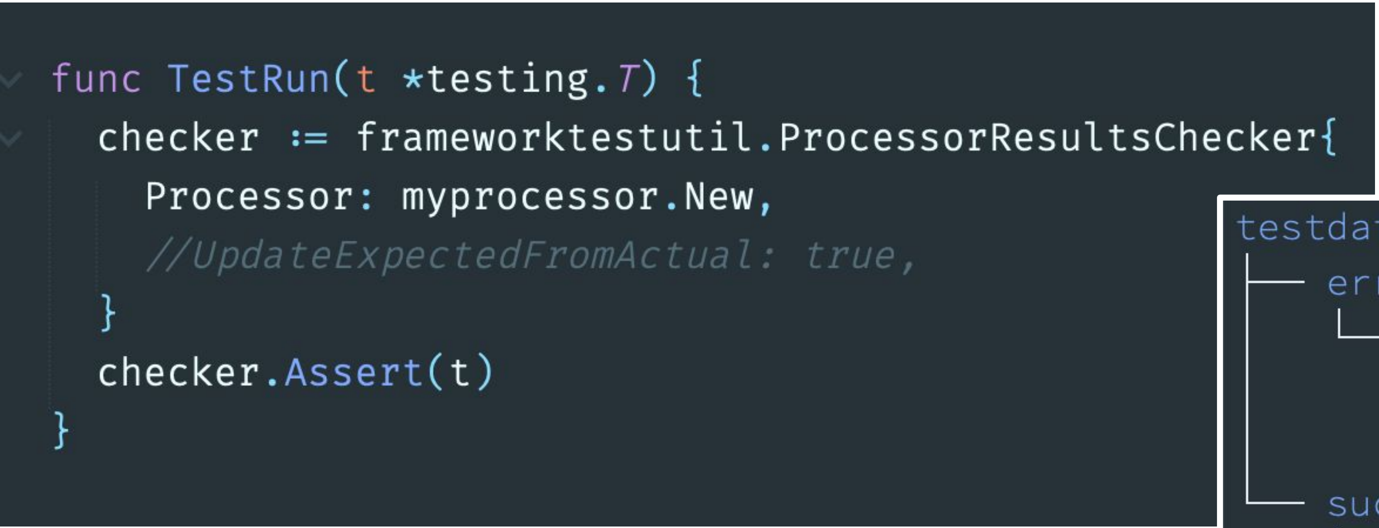
Best practices

Extension design

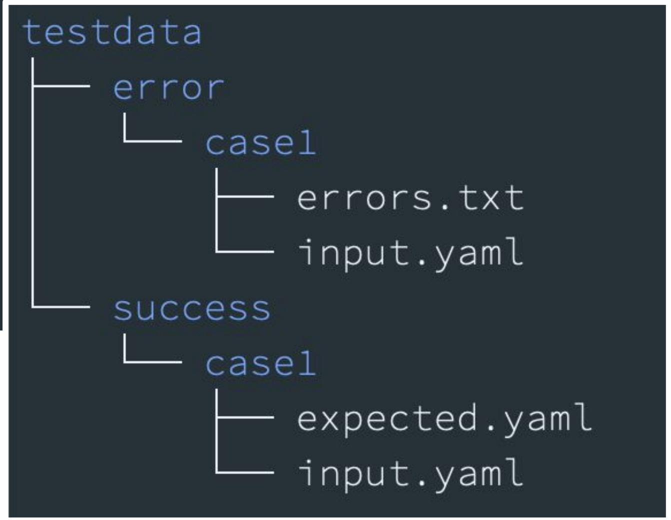
- Keep your extension **declarative**
- Make your extensions’ output **deterministic**
- Increment **API version** on changes that cause a diff

Best practices

Extension testing with fn framework

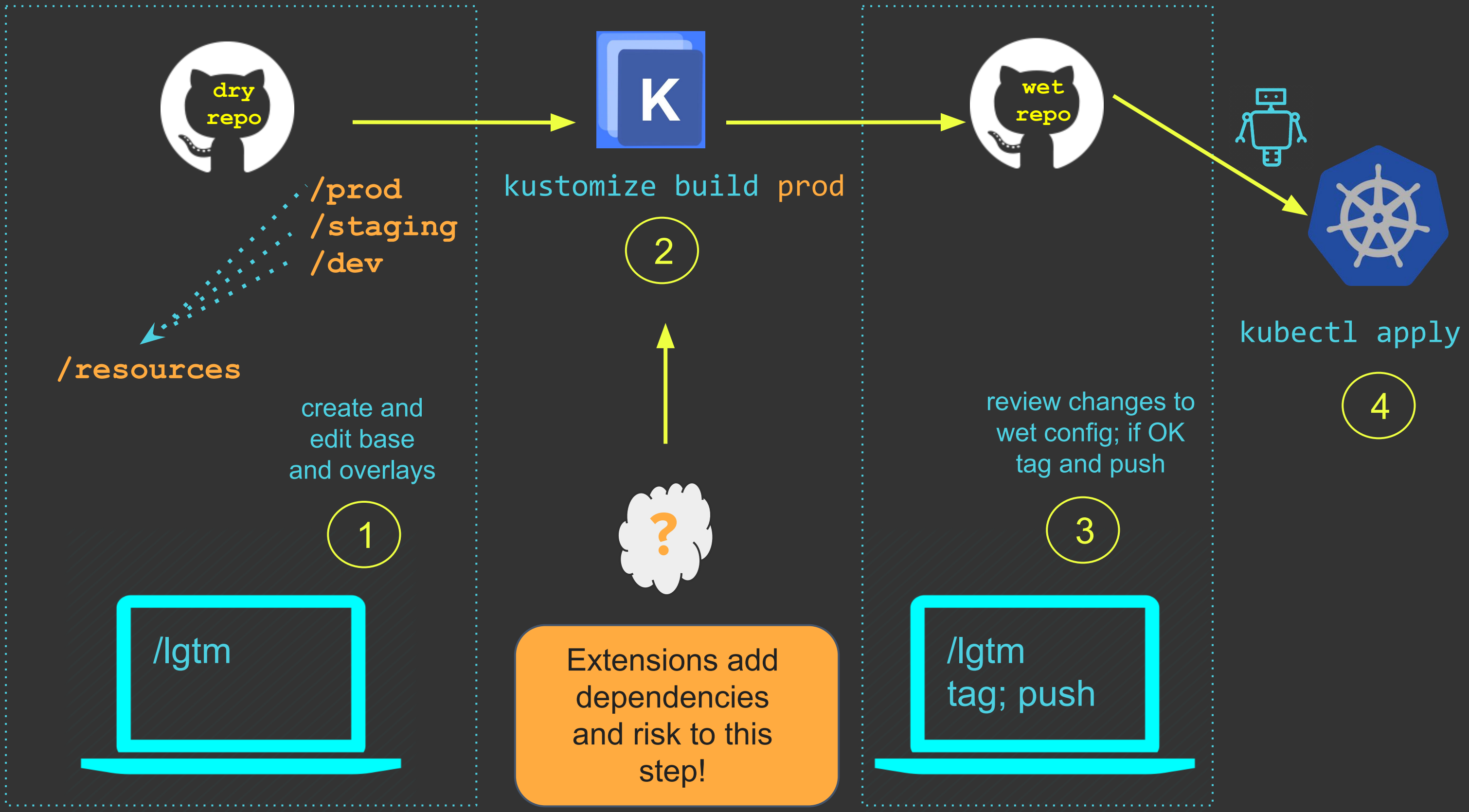


```go
func TestRun(t *testing.T) {
	checker := frameworktestutil.ProcessorResultsChecker{
		Processor: myprocessor.New,
		//UpdateExpectedFromActual: true,
	}
	checker.Assert(t)
}
```



```
testdata
├── error
│   └── case1
│       ├── errors.txt
│       └── input.yaml
└── success
    └── case1
        ├── expected.yaml
        └── input.yaml
```

Best practices: Use GitOps with extensions


dry repo
/prod
/staging
/dev
/resources
create and edit base and overlays
1
/lgtm
kustomize build prod
2
Extensions add dependencies and risk to this step!
wet repo
review changes to wet config; if OK tag and push
3
/lgtm
tag; push
kubectl apply
4

Resources

- Kustomize repo: sigs.k8s.io/kustomize
- Extensions docs: https://kubectl.docs.kubernetes.io/guides/extending_kustomize
- kyaml function framework docs: https://pkg.go.dev/sigs.k8s.io/kustomize/kyaml/fn/framework
- KEPs:
  - Kustomize Plugin Graduation: k/enhancements#2953
  - Kustomize Plugin Catalog: k/enhancements#2906
  - Kustomize Plugin Composition: k/enhancements#2299

RESILIENCE

REALIZED

KubeCon | CloudNativeCon

North America 2021