Ensuring Kubernetes manifests validity & compliance

A tooling overview

Yann Hamon
CNCF Meetup Berlin
Dec. 18, 2020

About me

- French, German
- Staff Engineer at @Contentful
- Interested in reliability, security and performance in distributed systems
- CoreDNS-Nodecache, Kube-secret-syncer, Kubeconform

But first...

Thank you Gareth!

- Author of or contributor to several of the tools presented here (Kubetest, Kubeval, Conftest, Rego rules for Kubesec checks)
- Keeping the resources those tools rely on up-to-date for many years
- Supportive & helpful

Gareth Rushgrove
garethr

800 followers · 11 following · 257
@snyk
Cambridge
gareth@morethanseven.net
morethanseven.net

A simple Nginx deployment

Copied from the Kubernetes documentation:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Associated service

I wrote this myself!!

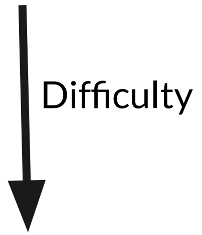
```
apiVersion: v1
kind: service
metadata:
   name: nginx-service
spec:
  ports;
  - port: 80
    protocol: tcp
  selector:
    run: nginx-service
```

Our mission, should you choose to accept it…

Use available, open-source tooling
to identify all the errors in this manifest

You will need to...

- Ensure the file is a valid YAML
- ... that is actually a valid Kubernetes manifest
- ... that conforms to your best practices
- ... that would actually apply without errors



YAML conformity

YAML validators / linters - yq, yamllint

Validating YAML - yq

- Multiple ways of doing so: yq, rq, python one-liners, …
- *yq* is a simple YAML validator

```
$ yq validate fixtures/nginx-service.yml
Error: yaml: line 7: mapping values are not allowed in this context
```

Validating / linting YAML: Yamllint

- Will also detect trailing spaces, inconsistent alignment, duplicate keys...
- Can recursively test folders

```
$ yamllint -d relaxed fixtures/
fixtures/nginx-service.yml
  5:6      error     trailing spaces  (trailing-spaces)
  7:9      error     syntax error: mapping values are not allowed here (syntax)
```

Yamllint

Pros:

- Very powerful and flexible
- Integrate with vim, emacs, more

Cons:

- YAML errors also covered by higher-level tools
- Not very fast
- Errors are mostly cosmetic

Fixed manifest

One validation error, one trailing whitespace

```
apiVersion: v1
kind: service 
metadata:
   name: nginx-service
spec:
  ports:
  - port: 80
    protocol: tcp
  selector:
    run: nginx-service
```

Learnings

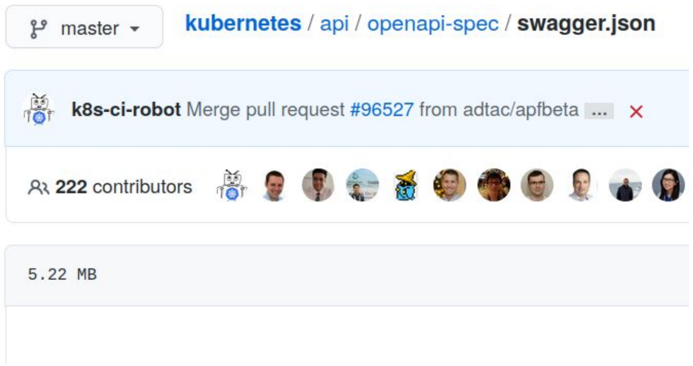
- Most higher-level tools will also validate YAML
- While consistent formatting is nice...
- Failing tests for whitespaces can be frustrating
- Use YAML linters in your editor

Kubernetes manifest conformity

Kubernetes validators - Kubeval, Kubeconform, Kubectl

Manifests should conform to K8S schemas

- Kubernetes publishes a Swagger file to describe its API
- Describes the Kinds of manifests, required/optional properties, types…
- Conforming to the schema is necessary but not sufficient for a manifest to be accepted by the API Server
- Especially important when upgrading to a new version of Kubernetes!


master
kubernetes / api / openapi-spec / swagger.json

k8s-ci-robot Merge pull request #96527 from adtac/apfbeta …

222 contributors

5.22 MB

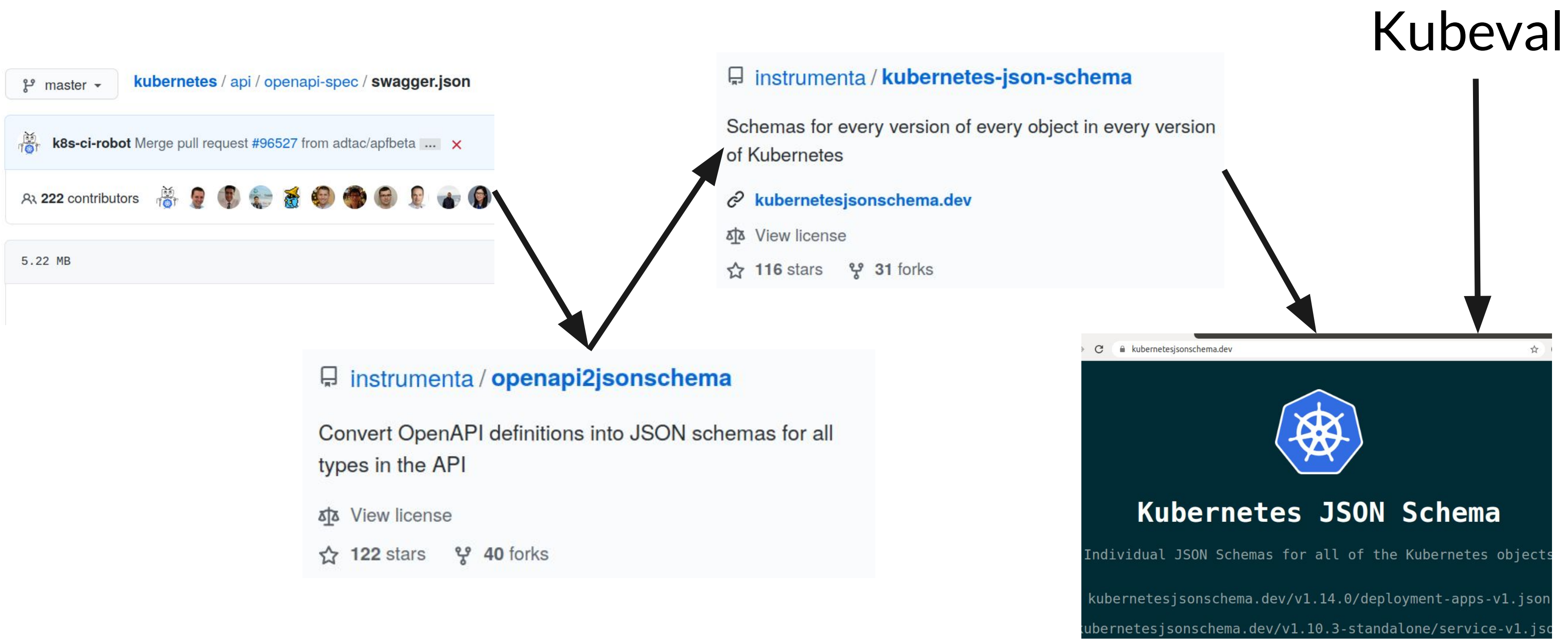
Validating a K8S manifest - Kubeval

```
$ kubeval --strict fixtures/nginx-deployment.yml fixtures/nginx-service.yml
PASS - fixtures/nginx-deployment.yml contains a valid Deployment
WARN - fixtures/nginx-service.yml contains an invalid service - kind: kind must be
one of the following: "Service"

$ echo $?
1
```

Simple enough. How does it work?

Kubeval's workflow



master

kubernetes / api / openapi-spec / swagger.json

k8s-ci-robot Merge pull request #96527 from adtac/apfbeta …

222 contributors

5.22 MB

instrumenta / openapi2jsonschema

Convert OpenAPI definitions into JSON schemas for all types in the API

View license

122 stars 40 forks

instrumenta / kubernetes-json-schema

Schemas for every version of every object in every version of Kubernetes

kubernetesjsonschema.dev

View license

116 stars 31 forks

kubernetesjsonschema.dev

Kubernetes JSON Schema

Individual JSON Schemas for all of the Kubernetes objects

kubernetesjsonschema.dev/v1.14.0/deployment-apps-v1.json

ubernetesjsonschema.dev/v1.10.3-standalone/service-v1.js

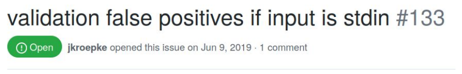
Kubeval

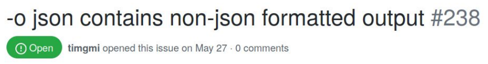
Pros:

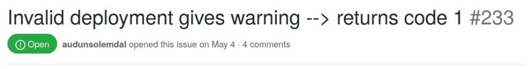
- Widely used
- Can iterate recursively over folders
- Can validate against different versions of Kubernetes

Cons:

- No support for Custom Resources*
- Some not-so-easy-to-fix bugs


validation false positives if input is stdin #133
Open jkroepke opened this issue on Jun 9, 2019 · 1 comment


-o json contains non-json formatted output #238
Open timgmi opened this issue on May 27 · 0 comments


Invalid deployment gives warning --> returns code 1 #233
Open audunsolemdal opened this issue on May 4 · 4 comments

Custom Resources support

- Welcome to the world of Kubernetes Operators
- Contentful uses the Prometheus, Jaeger, Kube-secret-syncer, and other operators… Custom Resources are everywhere
- One near-incident was due to missing validation of those files

Introducing Kubeconform

```
$ kubeconform -summary nginx-service.yml
Summary: 1 resource found in 1 file - Valid: 1, Invalid: 0, Errors: 0, Skipped: 0

$ echo $?
0
```

Kubeconform's CRD support

```
$ ./scripts/openapi2jsonschema.py \
https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/con
fig/crd/bases/sagemaker.aws.amazon.com_trainingjobs.yaml
JSON schema written to trainingjob_v1.json

$ ./bin/kubeconform -schema-location './{{ .ResourceKind }}{{ .ResourceAPIVersion
}}.json' fixtures/trainingjob-customresource.yaml

$ echo $?
0
```

Kubeconform

Pros:

- Feature-parity with Kubeval, same test-suite
- Fixes some outstanding Kubeval bugs
- Some performance improvements
- Flexible support for Custom Resources Schemas

Cons:

- Smaller community, not as battle-tested

Kubectl

```
$ kubectl apply --validate=true --dry-run=client -f fixtures/nginx-service.yml
service/nginx-service created (dry run)

$ echo $?
0
```

… the capitalisation error is not detected?

Kubectl #2

This time with a broken deployment…

```
$ kubectl apply --validate=true --dry-run=client -f fixtures/nginx-deployment-invalid.yml
error: error validating "fixtures/nginx-deployment-invalid.yml": error validating data:
ValidationError(Deployment.spec.template.spec.containers[0].ports[0].containerPort):
invalid type for io.k8s.api.core.v1.ContainerPort.containerPort: got "string", expected
"integer"; if you choose to ignore these errors, turn validation off with
--validate=false

$ echo $?
1
```

Kubectl

Pros:

- “Kubernetes upstream” project

Cons:

- Requires a connection to the Kubernetes cluster (even in client-side validation mode? #991)
- Unclear what is validated / poorly documented feature

Fixed manifest

“Service” needs to be capitalized

```
apiVersion: v1
kind: Service
metadata:
   name: nginx-service
spec:
  ports:
  - port: 80
    protocol: tcp
  selector:
    run: nginx-service
```

Learnings

- Use ~~Kubeval~~ Kubeconform to validate your manifests
- Similar functionality seems to be arriving in Kubectl
    - but poorly documented as of 2020
    - some validation errors not caught

Enforcing best practices / compliance

Testing Frameworks - Conftest, Kubesec

Best-practices, compliance

- “All deployments should have resource requests set”
- “All manifests should have a namespace explicitly defined”
- “No containers should run as privileged”
- “Resource names shall be lower-case”
- “Processes in containers should not run as root”
- [...]

Conftest, your manifests' policy framework

- Policies written in Rego - a language inspired by Datalog, itself derived from.. Prolog
- A declarative language - “Simpler & more concise” (.. supposedly)

“All manifests should have a namespace explicitly defined”

```
$ conftest test -p fixtures/policies/namespace.rego fixtures/nginx-deployment.yml
fixtures/nginx-service.yml
FAIL - fixtures/nginx-service.yml - no namespace set
FAIL - fixtures/nginx-deployment.yml - no namespace set

2 tests, 0 passed, 0 warnings, 2 failures

$ echo $?
1
```

“All manifests should have a namespace explicitly defined”

```
package main

kinds_to_skip = {
  "Apiservice",
  "Clusterrole",
  [...]
}

deny[msg] {
  not kinds_to_skip[lower(input.kind)]
  not input.metadata.namespace
  msg = "no namespace set"
}
```

Conftest

Pros:

- Powerful language
- Same language used for Admission controllers
- Quality of documentation
- A lot of existing rules
- Large community

Cons:

- Rego can be unfamiliar and “one more thing” to pick up

Security testing - Kubesec

Scores your manifests using a list of built-in rules

```
$ kubesec scan fixtures/nginx-*.yml
[...]
{
  "selector": "containers[] .securityContext .readOnlyRootFilesystem == true",
  "reason": "An immutable root filesystem can prevent malicious binaries being added to PATH and
increase attack cost",
  "points": 1
},{
  "selector": "containers[] .securityContext .runAsNonRoot == true",
  "reason": "Force the running image to run as a non-root user to ensure least privilege",
  "points": 1
},
[...]
```

Kubesec

Pros:

- Insightful rules
- Each rule is well-documented on their website with a description of potential impact (example)

Cons:

- Limited scope
- Check-out kubesec policies in Rego

Honourable mention: Kube-score

- Similar approach as Kubesec - scores manifests according to built-in rules
- Not extensible
- Good, documented list of checks

Honourable mention: Kube-linter

- Similar to kube-score, but will report errors instead of scores
- Not extensible
- In early stage of development

Honourable mention: Kubetest

- Testing Framework with tests written in Skylark, a subset of Python
- Archived in favour of Conftest

Fixed manifest

Set the namespace explicitly

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 80
    protocol: tcp
  selector:
    run: nginx-service
```

Learnings

- Conftest & OPA is the de-facto standard for policy-based testing
- Most people are not familiar with Rego, but it is not too hard to pick up
- It should be part of your CI!

Testing groups of resources

Featuring Conftest

Testing groups of resources

- “A deployment should specify a number of replicas, unless it has a matching HorizontalPodAutoscaling rule”
- “A service's 'selector' should have a matching deployment”
- “The configmap referenced by a deployment should exist”

All this is made possible by “conftest test --combine”

Warning: runtime complexity increases with the number of files tested

“A service's 'selector' should have a matching deployment”

```
package main

deployment[[input, file]] {
  input[deploymentfile].kind == "Deployment"
  input[deploymentfile].spec.selector.matchLabels.app == input[file].spec.selector.run
}

deny[msg] {
  input[file].kind == "Service"
  not deployment[[input, file]]

  msg := sprintf("Service %v points to non-existing deployment %v" ,
[input[file].metadata.name, input[file].spec.selector.run])
}
```

Now...

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: nginx~~-service~~
```

Does the manifest apply without errors?

Featuring Kubectl

Kubernetes & kubectl validate differently

Kubernetes' API, Controllers and Validation Webhooks add an extra layer of validation

```
$ kubectl apply --validate=true --dry-run=client -f fixtures/nginx-service.yml
service/nginx configured (dry run)

$ kubectl apply --validate=true --dry-run=server -f fixtures/nginx-service.yml
The Service "nginx" is invalid: spec.ports[0].protocol: Unsupported value: "tcp":
supported values: "SCTP", "TCP", "UDP"

$ echo $?
1
```

Fixed manifest

Protocol should be upper-case

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: nginx-service
```

Conclusion

Final manifest

```
apiVersion: v1
kind: Service    # Kubeconform, yamllint
metadata:
  name: nginx
  namespace: default # Conftest
spec:
  Ports:   # yq, yamllint
  - port: 80
    protocol: TCP  # Kubectl (dry-run=server)
  selector:
    run: nginx~~-service~~ # Conftest (across multiple files)
```

You should probably use...

- A YAML linter integrated with your editor
- Kubeconform to ensure your files are valid Kubernetes manifests
- Conftest with a library of policies for additional validation, best-practices and security compliance

Errors in manifests are easy to overlook

.. you should test your manifests!

- It is easy
- Save time by detecting errors earlier
- Kubernetes will not detect all mistakes - some might lead to operational problems
- Enforce best practices
- Ensure security compliance

Questions?