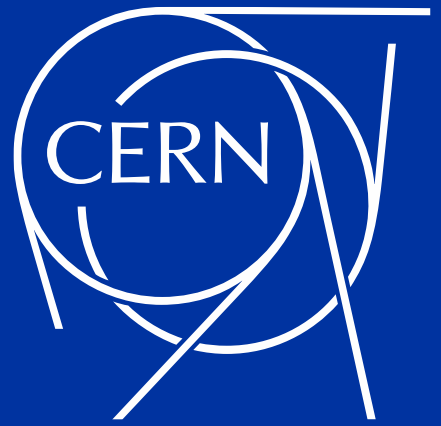


ORACLE

Migration between Kubernetes versions doesn't have to be error-prone?

Adrian Karasinski

27/03/2024

Service Mesh – static analysis of service dependencies

- We look at many different tools and things
- Part of the project was:
  - Investigation what we can use or do
  - Develop something together with ORACLE
- And we concluded that Kubernetes (K8s) is good target

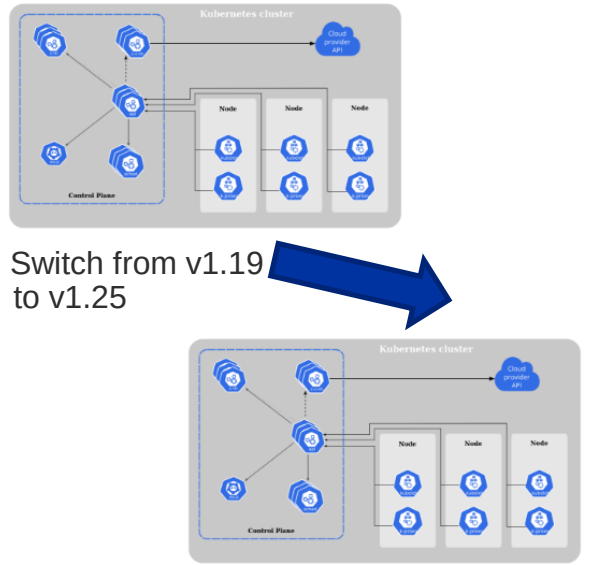
Specific case: Migration/Service Mesh in kubernetes

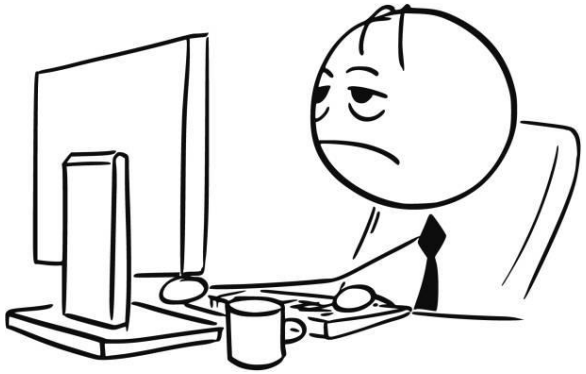
- Kubernetes is the main tool used in our CERN's group to application deployment
- Kubernetes simplifies, makes things faster and standardize application deployment
- Popular worldwide and in various industries
- Available in public clouds, private clouds, hybrid solutions, on-premise, etc.

Migration in K8s – current process of work

Typical step-by-step scenario:

- Manually checking release notes and should, but not usually no one does every templates (possibly thousands of files)
- Trying to re-deploy (re-use) all existing definitions in new cluster version
- Getting feedback only after everything has happened
- Unmeasured amount of work and verifications to be done, inestimable plans due to lack of feedback





Typical annual migration – to err is human

At least once per year:

1. a new version of the cluster has been announced
2. a few thousand definitions? Let's check it 1:1
3. any omission or unforeseen behavior: a chance that some critical systems will not work temporarily, which means abandoning live or professional plans ones in favor of maintenance.
4. **if we miss something in testing environment**, then most critical apps like Access Control, IMPACT, Electronic Document Handling may be unstable or be unavailable

A way to improve

Let's automate our work! Why bother a man when a machine can do it for him?

- automatically validate things
- feedback before,
  not surprise after

- on-the-fly conversion
- catch potential changes and divergences

A way to improve

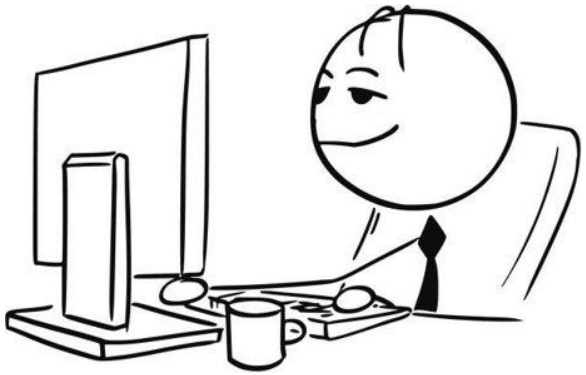
Let's automate our work! Why bother a man when a machine can do it for him?

- automatically validate things
- feedback before, not surprise after

- on-the-fly conversion
- catch potential changes and divergences

Our local, perfect world after the changes

- Daily work planned without surprises
- Reduced errors - you can spend time safely with friends or family, not on the phone or at the laptop
- More time doing smart things at work and less time reading schematics that machines/computers should read
- Let's leave creative work to people



Implemented tool

Technicalities:

- **51+ supported K8s kinds**
- **Full integration with K8s cluster API (kubeconf settings file)**
- **Integration also local templates/resources storage**
- **Simple deprecated/removed API versions view**

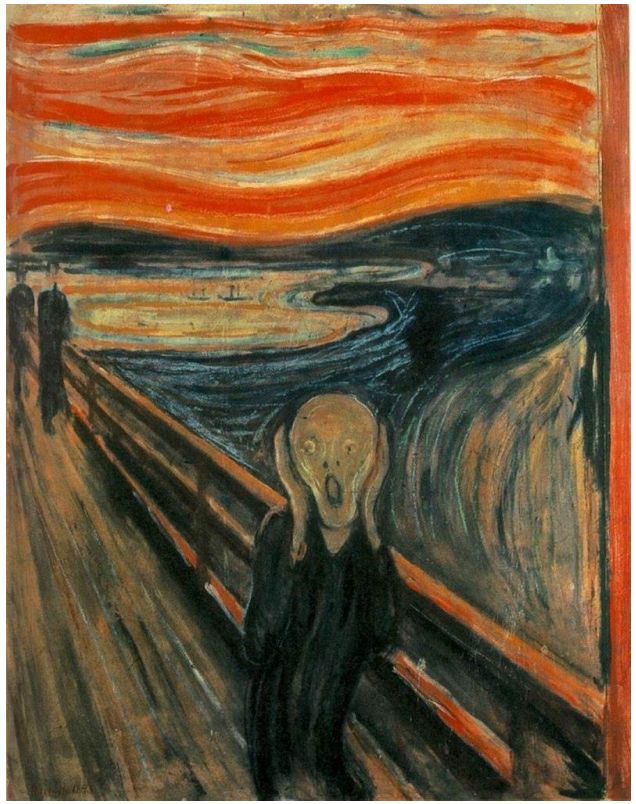
Testing

Challanges:

- The tool relies on OpenAPI schemas that we scrapped from kubernetes codebase
- Integration with the cluster and static files works great
- API deprecation spotting works great
- Scanning workloads usually works, but…

Plot twist - inconsistency

- As a part of the research it became apparent that there is inconsistency between schemas and kube-apiserver and various validation packages in K8s codebase.



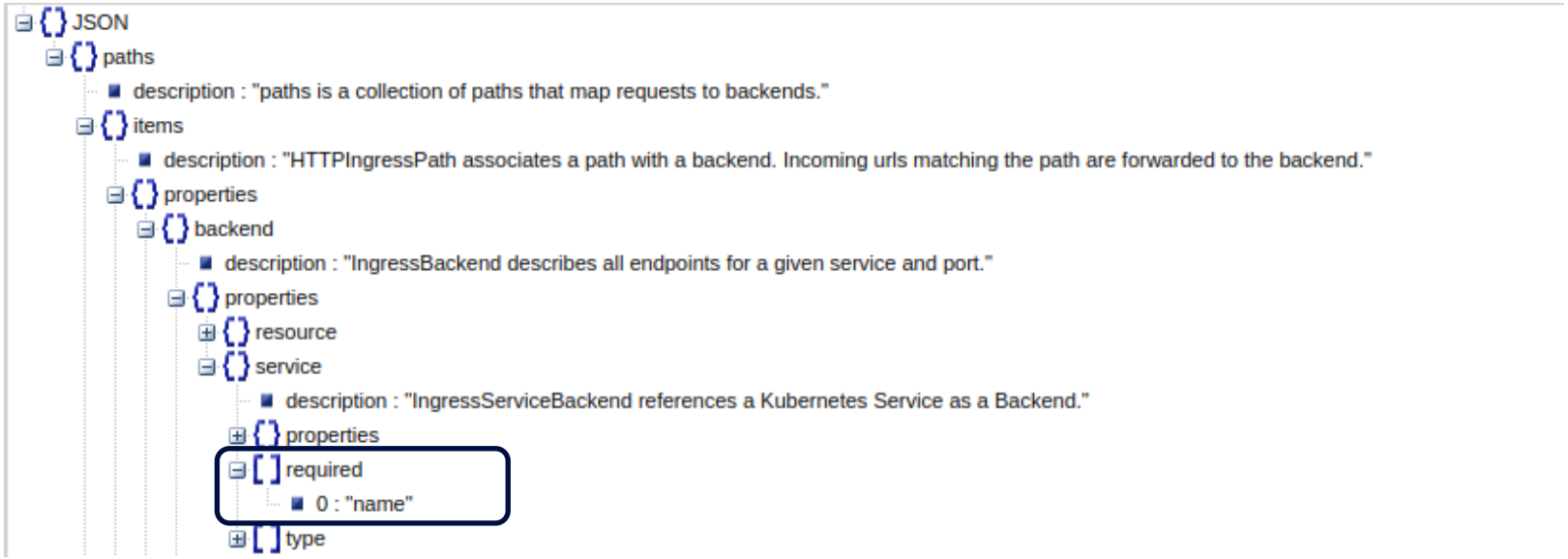
Inconsistency – codebase rules vs schema 1/2

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
    - http:
        paths:
          - path: /testpath
            pathType: Prefix
            backend:
              service:
                name: some-name
```

```
      } else if hasPortNumber {
        for _, msg := range validation.IsValidPortNum(int(backend.Service.Port.Number)) {
          allErrs = append(allErrs, field.Invalid(fldPath.Child("service", "port", "number"),
backend.Service.Port.Number, msg))
        }
      } else {
        allErrs = append(allErrs, field.Required(fldPath, "port name or number is required"))
      }
    default:
      allErrs = append(allErrs, field.Invalid(fldPath, "", "resource or service backend is required"))
    }
    return allErrs
}
```

The Ingress "minimal-ingress" is invalid: spec.rules[0].http.paths[0].backend: Required value: port name or number is required

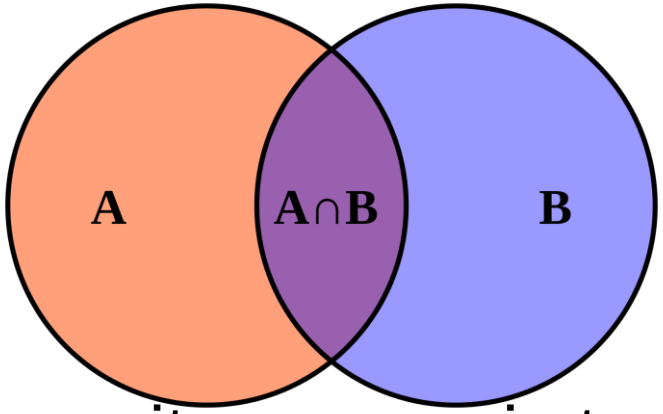
Inconsistency – codebase rules vs schema 2/2



The Ingress "minimal-ingress" is invalid: spec.rules[0].http.paths[0].backend: Required value: port name or number is required

Codebase

- Schema implementation is necessary but not sufficient condition to validate K8s workloads.



Legend: A - rules spotted into the OpenAPI schema, B - rules from the codebase

- It is impossible to write a generic tool, we could only target specific versions.
- Each successive version requires manual work, it is impossible to fully automate such process.

Global challenge

- There are multiple attempts in the Kubernetes community to address this problem, however none of them manage to handle the issue.
- It is impossible to write a generic tool, we could only target specific versions - each successive version requires manual work, it is impossible to fully automate such process.
- No official trace of the issue in the documentation.

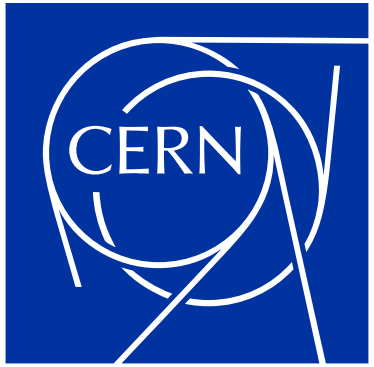
Conclusions

- Advertise/Alert the problem, so potential researchers and users will aware of this
- Spreading the issue to the wide K8s community and CNCF advisory groups
  - CNCF End User Technical Advisory Board (TAB)
  - TAG App Delivery

Conclusions

- Advertise/Alert the problem, so potential researchers and users will aware of this
- Spreading the issue to the wide K8s community and CNCF advisory groups
  - CNCF End User Technical Advisory Board (TAB)
  - TAG App Delivery

Q&A


CERN