Building and Maintaining a K8S Client Library

Stories from the trenches

Jussi Nummelin, Developer Advocate
@JNummelin

Me, myself and I

Name	Jussi Nummelin
Current Work	Developer / Advocate @ Kontena, Inc.
Previous	Tecnotree, Digia, Tieto, Nokia, …
Bio	All-around handyman on technical topics Working with containers & microservices for ~5 years Avid fly-fisher Hockey dad
Twitter / Github	@JNummelin / jnummelin

Topics

Why another client library?
Authentication is hard
POST vs. PUT vs. PATCH
Merge patching
Higher level constructs


PITFALLS
PITFALLS EVERYWHERE

Word of “wisdom”

Do not build your own client library, unless you really have to.
I mean REALLY!

Background

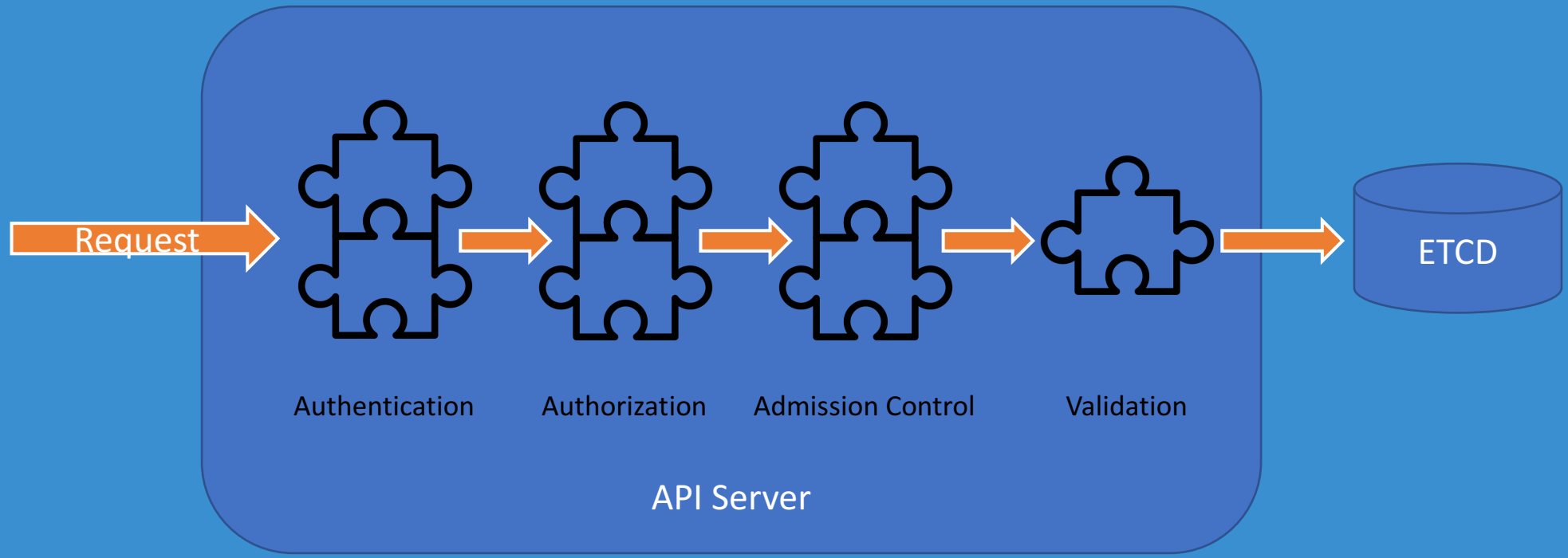
Pharos - A certified Kubernetes distro
~400 resources managed
Need for something lean&mean

Open source

github.com/kontena/k8s-client
github.com/kontena/pharos-cluster
github.com/kontena/mortar


SO IT BEGINS
memegenerator.net

Request processing stages



Auth is, ehem, hard

Auth is a moving target

API has 3 ways to auth
Kubeconfig has X number of ways
Often need to call external tool

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUd...
    server: https://35.228.64.213
  name: gke_kube-trial-224019_europe-north1-a_test-1-10-9
contexts:
- context:
    cluster: gke_kube-trial-224019_europe-north1-a_test-1-10-9
    user: gke_kube-trial-224019_europe-north1-a_test-1-10-9
  name: gke_kube-trial-224019_europe-north1-a_test-1-10-9
kind: Config
preferences: {}
users:
- name: gke_kube-trial-224019_europe-north1-a_test-1-10-9
  user:
    auth-provider:
      config:
        access-token: ya29.Glxk...
        cmd-args: config config-helper --format=json
        cmd-path: /Users/jussi/Downloads/google-cloud-sdk/bin/gcloud
        expiry: "2018-11-30T12:24:41Z"
        expiry-key: '{.credential.token_expiry}'
        token-key: '{.credential.access_token}'
      name: gcp
```

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBD...Qo=
    server: https://1.2.3.4:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJU...
    client-key-data: LS0tLS1CR...
```

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUd...
    server: https://xyz.yl4.eu-north-1.eks.amazonaws.com
  name: jussi-test-cluster.eu-north-1.eksctl.io
contexts:
- context:
    cluster: jussi-test-cluster.eu-north-1.eksctl.io
    user: jussi@jussi-test-cluster.eu-north-1.eksctl.io
  name: jussi@jussi-test-cluster.eu-north-1.eksctl.io
current-context: jussi@jussi-test-cluster.eu-north-1.eksctl.io
kind: Config
preferences: {}
users:
- name: jussi@jussi-test-cluster.eu-north-1.eksctl.io
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      args:
      - token
      - -i
      - jussi-test-cluster
      command: aws-iam-authenticator
      env: null
```

```yaml
apiVersion: v1
kind: Config
clusters:
  - name: cicd-cluster
    cluster:
      server: 'https://1.2.3.4:6443'
      certificate-authority-data: >-
        LS0tLS1CRUd...
users:
  - name: jussi
    user:
      token: 4eb21319fa...
contexts:
  - name: cicd-cluster
    context:
      user: jussi
      cluster: cicd-cluster
current-context: cicd-cluster
```

Kubeconfig is hard

KUBECONFIG can point to many files
Token may or may not be base64
Merging rules are complex
In-cluster vs. out of cluster

```
client = K8s::Client.autoconfig
```

POST vs. PUT vs. PATCH

Naïve approach

POST resource
GET resource
Merge with local (yaml)
PUT resource

Why not just PUT the resource?

Local yaml is **NOT** a full resource
API does not accept it
API has many admission controllers etc.

Solution

Record “last-applied-config”
“3-way” merge on client side
PATCH as “json-patch” type

Essentially a “kubectl apply -f …”

Merge patching

More than 1 way to PATCH

Strategic merge patch
JSON patch
JSON Merge patch

Strategic merge patch

Server side merge
Merge strategy decided on server side
So merging is different case-by-case

We use JSON patch

Diff ”last-applied-config” vs. New
Diff sent as ”application/json-patch+json”
RFC6902

```
[
  {
    "op": "remove",
    "path": "/metadata/labels/foo"
  },
  {
    "op": "add",
    "path": "/metadata/labels/bar",
    "value": "foobar"
  },
  {
    "op": "add",
    "path": "/spec/containers/0/ports/1",
    "value": {
      "name": "bar",
      "port": 90,
      "targetPort": 90
    }
  }
]
```

Empty vs. null vs. undefined

They are the same...

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: docker.io/nginx/:1-alpine
    resources:
      limits:
        memory: "10Mi"
        cpu: "100m"
```

==

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels: {}
spec:
  containers:
  - name: myapp
    image: docker.io/nginx/:1-alpine
    resources:
      limits:
        memory: "10Mi"
        cpu: "100m"
    ports: []
```

...until they are not

```
$ kubectl patch deployment myapp --type='json' -p='[{"op": "add",
"path":"/spec/tolerations", "value":[]}]'
deployment.extensions/myapp patched

$ kubectl patch deployment myapp --type='json' -p='[{"op": "replace",
"path":"/spec/tolerations", "value":[]}]'
Error from server: jsonpatch replace operation does not apply: doc is missing
key: /spec/tolerations
```

Empty might be meaningful!?

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from: []
```

With JSON patch

Drop ”empty” values from original
or
Deal with ”empty” values during diff

API Discovery & CRDs

API discovery

Map resource types to API paths
CRDs
Static vs. Dynamic
Caching?

CRDs

CRDs are hard with many libs
Generate resource types?
Un-typed resources → CRDs are easy

```
$  irb -r wirb --inspect wirb
2.5.3 :001 > require 'k8s-client'
 => true
2.5.3 :002 > client = K8s::Client.autoconfig; nil
 => nil
2.5.3 :003 > cluster_info = client.api("crd.projectcalico.org/v1").
2.5.3 :004 >   resource("clusterinformations").
2.5.3 :005 >   get("default"); nil
 => nil
2.5.3 :006 > cluster_info.metadata.name
 => "default"
2.5.3 :007 > cluster_info.metadata.spec.calicoVersion
Traceback (most recent call last):
        2: from /Users/jussi/.rvm/rubies/ruby-2.5.3/bin/irb:11:in `<main>'
        1: from (irb):7
NoMethodError (undefined method `calicoVersion' for nil:NilClass)
2.5.3 :008 > cluster_info.spec.calicoVersion
 => "v3.3.6"
```

Thank you!