# Self-service web hosting made easy with containers and Kubernetes Operators

CERN WebServices' experience with Kubernetes extensibility

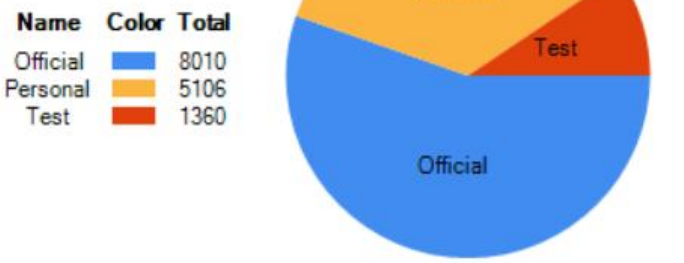
**Alex Lossent**

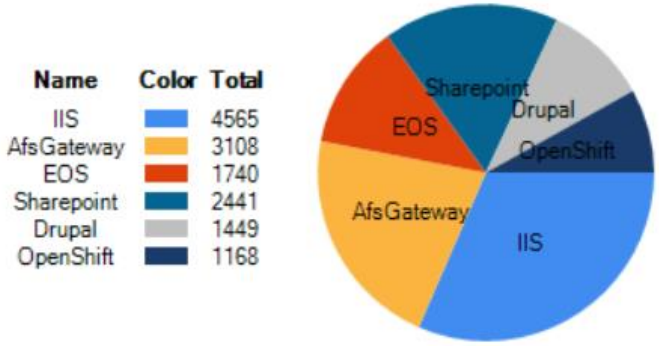
**CERN - IT-CDA-WF**

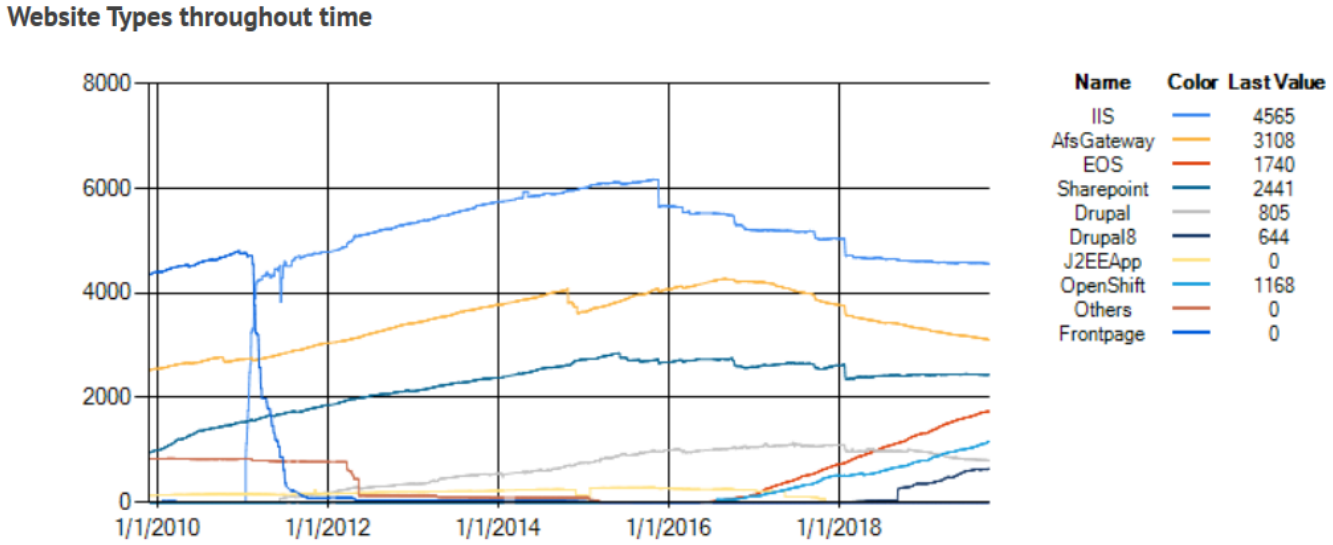
# Outline

- Context: web hosting at CERN
- The Kubernetes Operator pattern
- Operators for self-service site provisioning
- Operators for infrastructure deployment
- Operator Lifecycle Manager

# Context: Web Hosting at CERN



# Web Hosting strategy

- **Strategy: consolidate web hosting on containers**
  - Platform-as-a-Service (PaaS) scenario
    - Heroku-style custom app hosting (node.js, Java, Rails etc.)
    - Self-service application templates (Jenkins, Discourse etc.)
  - Static+CGI (WebEOS)
  - Content Management Systems (Drupal, Twiki, MkDocs etc.)

# How to provision and manage sites on OKD?

- **Different site types are deployed differently**
  - PaaS: one namespace/app, delegate full permissions to owner (within quota)
  - CMS (e.g. Drupal): one namespace/app, controlled environment with few permissions delegated to site owner
  - Static content (e.g. WebEOS): single deployment serving many sites
- **Desiderata:**
  - Minimal effort to develop a UI
  - Minimal effort to develop an API
  - Minimal coupling between various site types
  - High level of component reusability (DNS integration, SSO registration etc.)

# Kubernetes Operator pattern

- **Operator = Kubernetes native application**
  - Deployed on Kubernetes
  - Managed using the Kubernetes API and CLI tooling
- **Automate management of containerized applications**
  - Leveraging Kubernetes extensibility
- **References:**
  - https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
  - https://coreos.com/blog/introducing-operators.html
  - https://coreos.com/operators/

# Managing web site provisioning with Kubernetes API and CLI

- **CRD = Custom Resource Definition**

```yaml
kind: DrupalSite
apiVersion: drupal.cern.ch/v1
metadata:
  name: mysite

spec:
  owner: myuser
  drupalVersion: 8.7.0

status:
  currentOwner: myuser
  currentDrupalVersion: 8.7.0
```

Desired state

Information on current state

# Implementing provisioning logic

Site owner

Create (via Kubernetes API)

```
kind: DrupalSite
apiVersion: drupal.cern.ch/v1
metadata:
  name: mysite

spec:
  owner: myuser
  drupalVersion: 8.7.0

status:
  currentOwner: myuser
  currentDrupalVersion: 8.7.0
```

1. Watch resource changes

2. Compare desired state with current state of the world

Kubernetes API

**Reconciliation loop**

3. Apply changes to the world

External services

- Kubernetes namespace
- Kubernetes deployment
- Set up Database
- DNS registration
- ….

4. Report in status section

Containerized application (typically written in Go)

# Change workflow

Site owner

Update (via Kubernetes API)

```
kind: DrupalSite
apiVersion: drupal.cern.ch/v1
metadata:
  name: mysite

spec:
  owner: myuser
  drupalVersion: 8.8.0

status:
  currentOwner: myuser
  currentDrupalVersion: 8.8.0
```

1. Watch resource changes

2. Compare desired state with current state of the world

Kubernetes API

**Reconciliation loop**

3. Apply changes to the world

External services

- Backup database
- Migrate database
- Update Docker image
- Check site works
- (rollback if not)

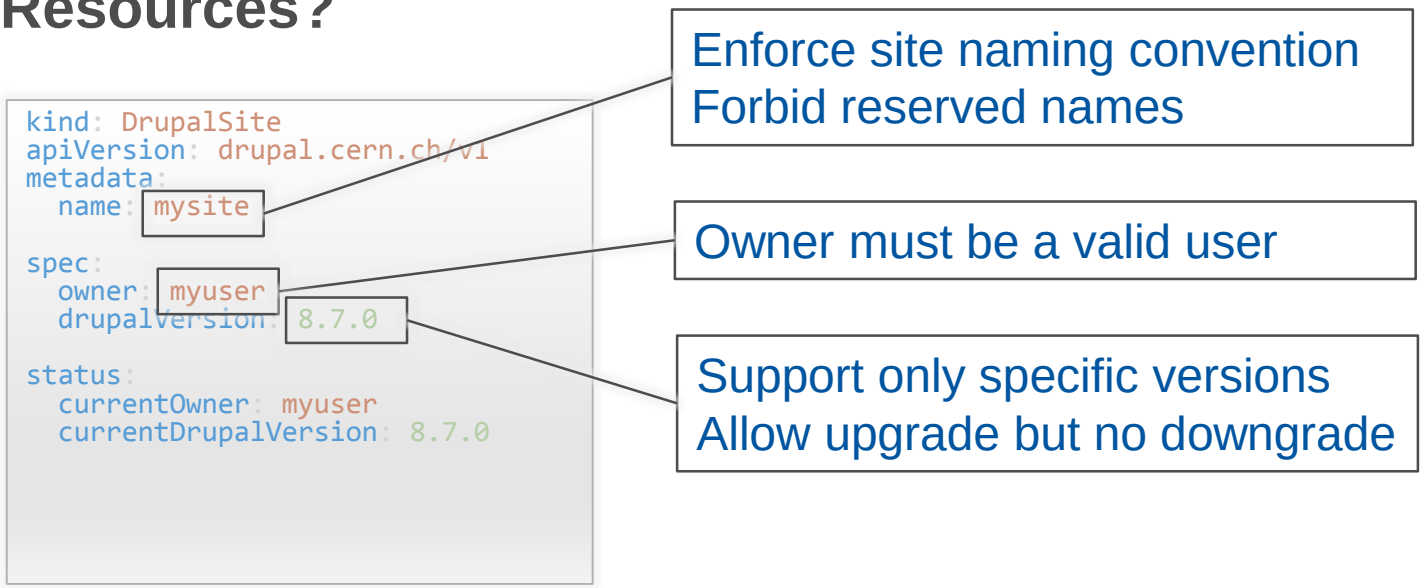
4. Report in status section

# Benefits

- **Kubernetes services for CRD:**
  - API, CLI (kubectl/oc)
  - Authentication, authorization, RBAC
  - Event notifications (watches)
  - Resource lifecycle management
  - Auto-generated UI in Openshift web console
- **Single reconciliation loop**
  - (De-)provisioning
  - Correcting any divergence (e.g. periodic loop execution)
  - State transitions (e.g. upgrade Drupal 8.7.0 -> 8.8.0, change owner)

# Interlude: constraints and policies

- **How to enforce constraints and policies on Custom Resources?**




- **Kubernetes offers “Dynamic Admission Control”**
  - Calls custom admission web hooks that validate resources

# Open Policy Agent

invariants.rego — ~/src/policies/admission-control/kubernetes

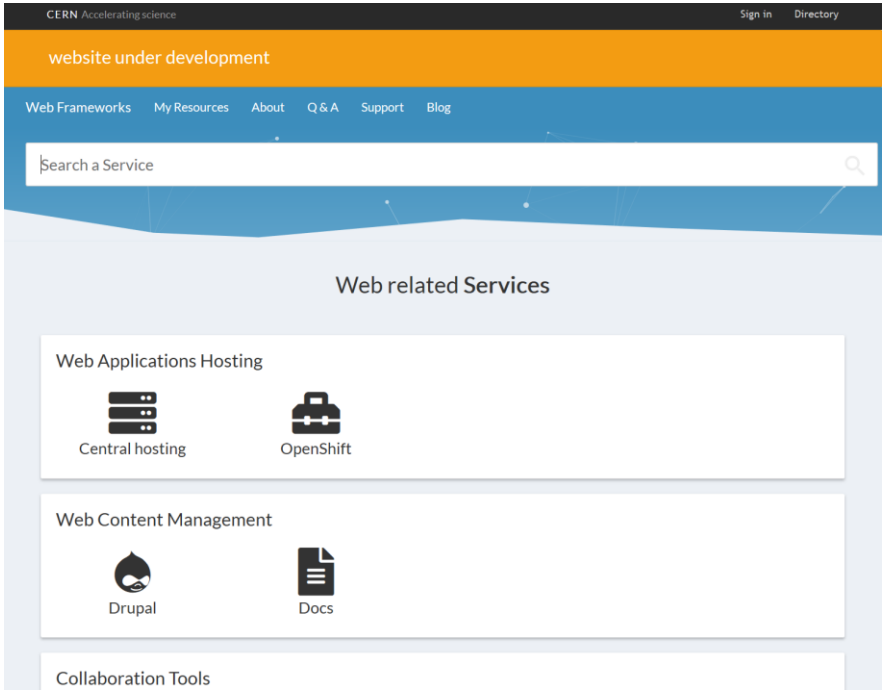
```
# Kubernetes Admission Control Invariants

package kubernetes.invariants

import data.kubernetes.ingresses
import data.kubernetes.namespaces

# ------------------------------------------------------------------
# Ingress Invariants

# Generates a list of non-compliant ingresses identified by `namespace`
# and ingress specification `name`.
violations[{
    "namespace": namespace,
    "name": name,
    "message": "ingress hostname must match whitelist"
}] {
    ingress := ingresses[namespace][name]
    host := ingress.spec.rules[_].host
    not contains(whitelist[namespace], host)
}

# Generates a list of allowed hostnames per namespace.
whitelist[namespace] = hosts {
    obj := namespaces[namespace]
    annotations := obj.metadata.annotations
    annotation := annotations["acmecorp.com/hostname-whitelist"]
    hosts := json.unmarshal(annotation)
}

# ------------------------------------------------------------------
# Helpers

# Checks if `list` includes an element matching `item`.
contains(list, item) {
    list[_] = item
}
```

- **Generic policy framework**
  - Declarative language: “rego”
  - Open-source CNCF project
- **Implement constraints and policies easily**
  - Well integrated with Kubernetes
- **Keeps operator logic focused on the reconciliation loop**

Source: https://www.openpolicyagent.org/

# How do end-users actually provision sites?

- **Dedicated CRD + Operator for each site type**
  - DrupalSite, WebEosSite, PaasSite…
- **API and CLI readily available**
- **Single-page-application front-end for a friendly UI**



# Automating common integration tasks

- **Dedicated Operators to automate common sub-tasks**
  - E.g. SsoRegistration, DnsRegistration, MatomoStatistics…
  - Parent/child relationships between Kubernetes resources

```yaml
kind: MatomoStatistics
apiVersion: matomo.cern.ch/v1
metadata:
  name: mysite

spec:
  host: mysite.web.cern.ch
  owner: myaccount

status:
  siteId: 1234
  statsUrl: https://matomo.cern
```

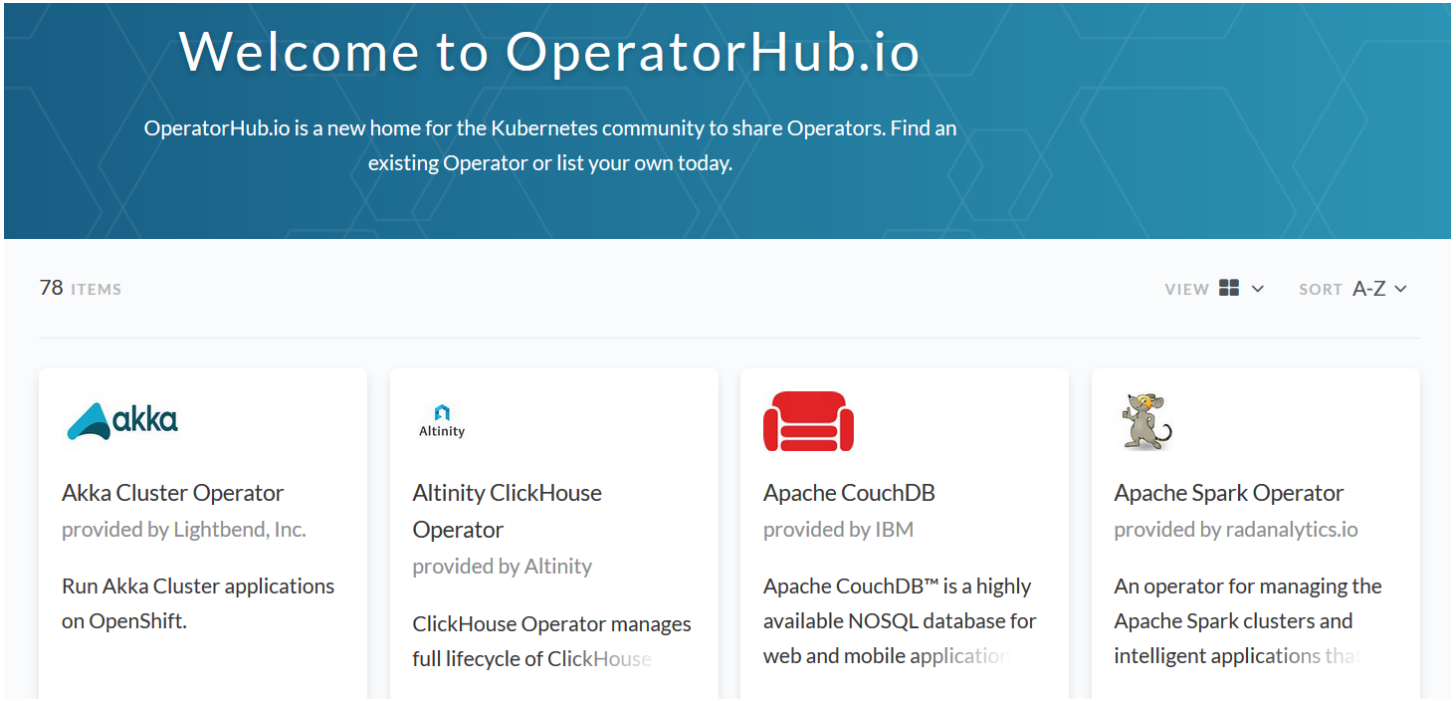
```yaml
kind: SsoRegistration
apiVersion: oauth.cern.ch/v1
metadata:
  name: mysite

spec:
  redirectUri: https://mysite.web.cern.ch/oauth

status:
  appId: 1234abcd
  appSecretName: oauth-secret
```

# Operator development

- **Operator Framework**
  - https://github.com/operator-framework/operator-sdk
- **Operator SDK provides all the plumbing**
  - Watching events, caching state of the world, retrying operations etc.
  - Primary language: Go
- **Operators can also be generated automatically from:**
  - Ansible playbooks (all-purpose configuration management tool)
  - Helm (the “package manager” for Kubernetes)

# Operator Hub

- **Operators quickly becoming an industry standard**



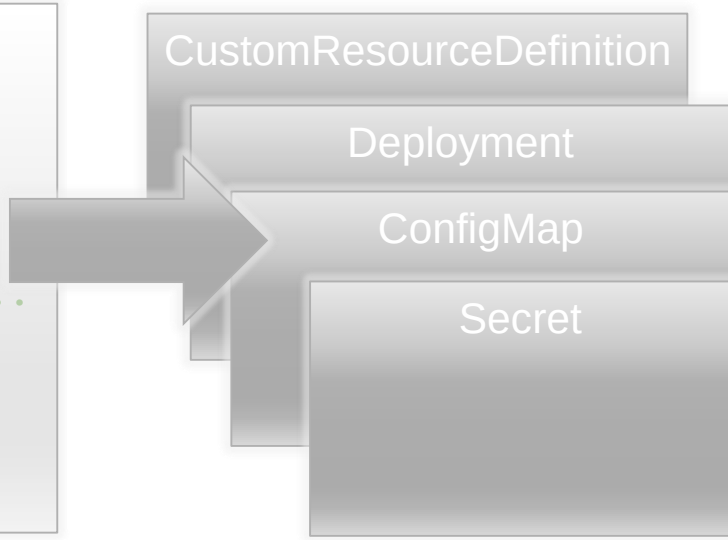
# Operators for infrastructure deployment

- **Openshift 4 design goal: provide full-stack automation with operators**
  - Infrastructure provisioning (AWS, Openstack etc.)
  - OS updates (RedHat/Fedora CoreOS)
  - Openshift/Kubernetes platform
  - Integrated services (ingress routers, monitoring, application catalogs etc.)

# Operators to deploy our own infrastructure components

- **Our custom operators are Kubernetes applications themselves**
  - DrupalSite, DnsRegistration, SsoRegistration etc.
- ***We can automate their installation, configuration & upgrades… with operators!***
  - e.g. auto-generated from a Helm chart

```yaml
kind: CernSsoIntegrationComponents
apiVersion: deploy.cern.ch/v1
metadata:
  name: prod

spec:
  ssoApiEndpoint: https://authsvc.cern.ch/api/...
  ssoApiAuthToken: ...

status:
  deploymentStatus: OK
```



# Operator Lifecycle Manager

- **How to distribute, update, monitor operator deployment?**
  - Or manage dependencies?
- **OLM provides a framework where:**
  - Operators are packaged and published via a catalogue and *channels* (e.g test, staging, prod…)
  - Subscription to channels automates the process of discovering updates and deploying them in a cluster
- **The OLM is itself… An operator!**
  - CRDs to define catalogs, channels, operators to deploy etc.

# Conclusion

- **A work in progress**
  - Finalizing infrastructure components
  - Developing operators for the various site types
  - Targeting deployment together with upgrade to Openshift 4
- **Very good experience with Kubernetes extensibility features**
  - Operators shine at automating tasks that previously required significant operational knowledge and manual intervention

# Questions?