Effective management of Kubernetes resources

GitOps for cluster admins

Tom Coufal
Principal Software Engineer, Red Hat
What we’ll discuss today

▸ Cluster lifecycle and role of Cluster Ops
▸ Experience the chaos
▸ Let’s talk YAML
▸ Bring order to chaos
Operations

Three graces of OPS in cloud

Resource provisioning → Cluster management → Application maintenance
Operations

Three graces of OPS in cloud

- Resource provisioning
- Terraform
- Ansible
- Kubernetes
Operations

Three graces of OPS in cloud

Kustomize.io
HELM
argo
flux

Application maintenance
Operations

Three graces of OPS in cloud

Cluster management
Cluster management

What exactly does that mean?

1. Multi-tenancy ➔ Namespaces
2. Cluster upgrades ➔ Cluster Roles
3. Storage management ➔ Custom Resource Definitions
4. Network management ➔ Storage Classes
   ➔ Operators
Kubernetes resources are declarative

Designed to be human readable

Client side organization is totally up to user

---
apiVersion: v1
kind: Namespace
metadata:
  name: sovereign-cloud
spec: {}
---

apiVersion: 2023.fosdem.org/v1
kind: Talk
metadata:
  name: effective-management-of-resources
  namespace: sovereign-cloud
  annotations:
    full-name: Effective management of Kubernetes resources for cluster admins
spec:
  speaker: tumido
...
status:
  phase: HappeningNow
---

apiVersion: 2023.fosdem.org/v1
kind: Speaker
metadata:
  name: tumido
spec:
  ...
Kubernetes client doesn’t impose any layout restrictions. All is client specific.

Bash `**/*` is the limit.
Emphasis on easy and quick deployment

Per-environment `values.yaml` file

No layout requirements

No specific entrypoint
# base/tracks/sovereign-cloud.yaml
1. apiVersion: v1
2. kind: Namespace
3. metadata:
4.   name: sovereign-cloud
5. spec: {}

# base/talks/effective-management-of-resources.yaml
1. apiVersion: 2023.fosdem.org/v1
2. kind: Talk
3. metadata:
4.   name: effective-management-of-resources
5.   namespace: sovereign-cloud
6.   annotations:
7.     full-name: Effective management of Kubernetes resources for cluster admins
8. spec:
9.   speaker: tumido

# base/kustomization.yaml
1. apiVersion: kustomize.config.k8s.io/v1beta1
2. kind: Kustomization
3. resources:
4.   - tracks/sovereign-cloud.yaml
5.   - talks/effective-management-of-resources.yaml

# overlays/prod/kustomization.yaml
1. apiVersion: kustomize.config.k8s.io/v1beta1
2. kind: Kustomization
3. resources:
4.   - ../../../base
Transparency

🚫 Build our own solution, build your own CI/CD solution
✓ Use established project in OSS space, fully auditable
Configuration stability

**❌ Monolithic configuration for whole cluster fleet**

**✅ Git blame works, unit testable, rollback per cluster**
File Mapping

❌ Multiple Kubernetes resources in a single YAML file

✅ One file represents ONLY one Kubernetes object definition
Direct definitions

- Object definition properties can be templated
- Each file is readable without processing
No duplicity

❌ Keep a separate copy of resource definition per environment

✅ Definitions are reused and referenced instead of copied
Clarity

File names vaguely descriptive of its content

Name of each file technically describes its content
Every resource deployed has a definition here.

BASE

```
├── base
│   ├── core
│   │   └── namespaces
│   │       └── sovereign-cloud.yaml
│   ├── 2023.fosdem.org
│   │   └── talks
│   │       └── effective-management-of-resources.yaml
│   └── speakers
│       └── tumido.yaml
│
│
│
│
│
│
│

├── overlays
│   ├── clusterA
│   │   └── kustomization.yaml
│   ├── clusterB
│   │   ├── speakers
│   │   │   └── tumido_patch.yaml
│   │   └── kustomization.yaml
│   └── clusterC
│       └── kustomization.yaml

OVERLAYS

Each overlay represents a cluster environment.

base/<API_GROUP>/<KIND>/<NAME>
Can be reused for each manifest in base separately
e.g. Add a RoleBinding for each Namespace

COMPONENTS

BUNDLES
Compose functional couplings of base manifests into a single unit
e.g. Add all resources that installs a Cert Manager
# overlays/moc/smaug/kustomization.yaml

1. apiVersion: kustomize.config.k8s.io/v1beta1
2. kind: Kustomization
3. resources:
4. - ../common
5. - ../../../base/apiextensions.k8s.io/customresourcedefinitions/
   prowjobs.prow.k8s.io
6. - ../../../base/core/namespaces/prow
7. - ../../../base/rbac.authorization.k8s.io/clusterroles/node-labeler
8. - ../../../base/rbac.authorization.k8s.io/clusterrolebindings/node-labeler
9. - ../../../bundles/cert-manager
10. - clusterversion.yaml
11. patches:
12. - groups/cluster-admins.yaml

Cluster overlay
Cluster specific configuration
Uses region’s shared overlay common
Imports bundles
Patches base resources
Conclusion

Evaluate, review, retrospect

+ No definition duplicity
+ Manifest clarity and readability
+ No manifest confusion
+ Clear, small, effective set of rules
+ Easy CI/CD
+ Unit tests (per bundle)
+ Integration tests (per cluster)

- Boilerplate bloat and fatigue
- Kustomization complexity and confusion
- Patch complexity
- Manifests in base can be partials (limits static scheme validation)
Addopters

Lessons learned, lessons shared
Resources

Operate First
operate-first.cloud

Where’s the Docs?
service-catalog.operate-first.cloud/adrs

Where’s the Source
github.com/operate-first/apps/tree/master/cluster-scope
THX & Q & A

github.com/tumido

fosstodon.org/@tumido

twitter.com/tumido_