Effective management of Kubernetes resources

GitOps for cluster admins

Tom Coufal

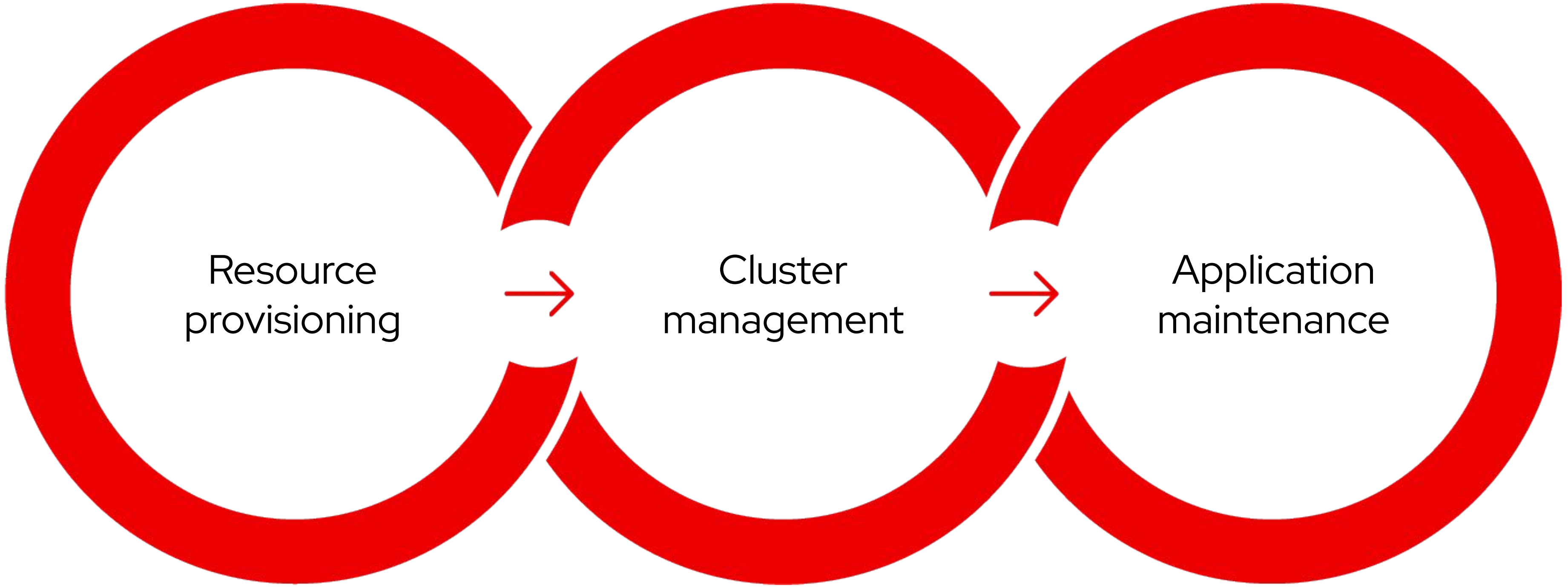
Principal Software Engineer, Red Hat

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



Operations

Three graces of OPS in cloud

Resource provisioning

HashiCorp Terraform

ANSIBLE

Operations

Three graces of OPS in cloud

Kustomize.io

HELM

argo

flux

Application maintenance

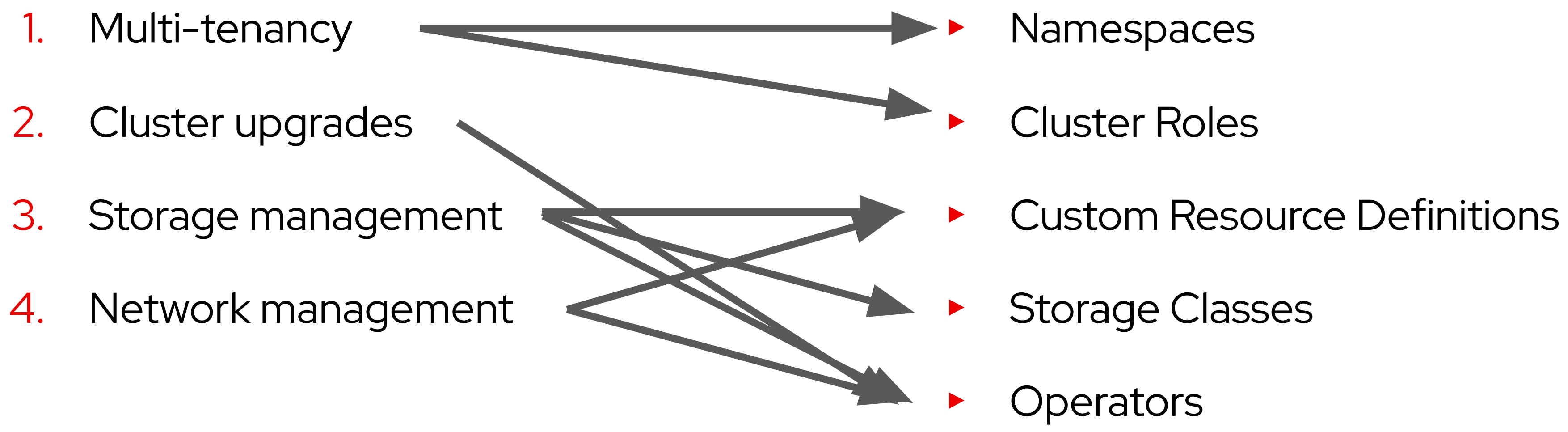
Operations

Three graces of OPS in cloud

?
Cluster management
?

Cluster management

What exactly does that mean?



1. Multi-tenancy
2. Cluster upgrades
3. Storage management
4. Network management

- Namespaces
- Cluster Roles
- Custom Resource Definitions
- Storage Classes
- Operators

```yaml
---
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:
  ...
```

YAML chaos

Kubernetes resources are declarative

Designed to be human readable

Client side organization is totally up to user

```
├── src/...
├── docs/...
└── [manifests|deploy]
    ├── 0_crds.yaml
    ├── 0_namespace.yaml
    ├── 1_speaker.yaml
    └── 1_talk.yaml
```

```
├── src/...
├── docs/...
└── [manifests|deploy]
    ├── dev.yaml
    ├── stage.yaml
    └── prod.yaml
```

```
├── src/...
├── docs/...
└── [manifests|deploy]
    ├── dev
    │   ├── 0_crds.yaml
    │   ├── 0_namespace.yaml
    │   ├── 1_speaker.yaml
    │   └── 1_talk.yaml
    ├── stage
    └── prod
```

Chaos as files

Kubernetes client doesn’t impose any layout restrictions.

All is client specific

Bash ****/*** is the limit

```
# templates/2023_fosdem.yaml

{{- range .Values.tracks }}
---
apiVersion: v1
kind: Namespace
metadata:
   name: {{ .name }}
spec: {}
{{- $trackName := .name }}
{{- range .talks }}
---
apiVersion: 2023.fosdem.org/v1
kind: Talk
metadata:
  name: {{- .name }}
  namespace: {{- $trackName }}
  annotations:
    full-name: {{- .fullName }}
spec:
  speaker: {{- .speaker }}
{{- end }}
{{- end }}
```

```
# values.yaml

tracks:
- name: sovereign-cloud
  talks:
  - name: effective-management-of-resources
    fullName: ...
    speaker: tumido
  ...
...
```

HELM

Emphasis on easy and quick deployment

Per-environment **`values.yaml`** file

No layout requirements

No specific entrypoint

Red Hat

# base/tracks/sovereign-cloud.yaml

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

# base/talks/effective-management-of-resources.yaml

```yaml
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
```

# base/kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- tracks/sovereign-cloud.yaml
- talks/effective-management-of-resource...
```

# overlays/prod/kustomization.yaml

```yaml
apiVersion: kustomize.config...
kind: Kustomization
resources:
- ../../base
```

Kustomize.io

Not a templating engine

Has layout requirements:
base/overlays concept

Encourages code reuse will complain in case of multiple definitions

Red Hat

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**

```
├── 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
```

`base/<API_GROUP>/<KIND>/<NAME>`

BASE

Every resource deployed has a definition here

OVERLAYS

Each overlay represents a cluster environment

```
├── base
│   ├── core
│   │   └── namespaces
│   │       └── sovereign-cloud
│   │           ├── namespace.yaml
│   │           └── kustomization.yaml
│   └── 2023.fosdem.org
│       ├── talks
│       │   └── effective-management-of-resources
│       │       ├── talk.yaml
│       │       └── kustomization.yaml
│       └── speakers
│           └── tumido
│               ├── speaker.yaml
│               └── kustomization.yaml
├── components
│       └── cfp-deadline
├── bundles
│       └── sovereign-cloud-full-track
│           └── kustomization.yaml
└── overlays
    ├── regionA
    │   └── clusterA
    └── regionB
        ├── common
        ├── clusterB
        └── clusterC
```

COMPONENTS

Can be reused for each manifest in base separately

e.g. Add a **`RoleBinding`** for each **`Namespace`**

BUNDLES

Compose functional couplings of base manifests into a single unit

e.g. Add all resources that installs a Cert Manager

```yaml
# overlays/moc/smaug/kustomization.yaml

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../common

- ../../../../base/apiextensions.k8s.io/customresourcedefinitions/
                                                        prowjobs.prow.k8s.io

- ../../../../base/core/namespaces/prow

- ../../../../base/rbac.authorization.k8s.io/clusterroles/node-labeler
- ../../../../base/rbac.authorization.k8s.io/clusterrolebindings/node-labeler

- ../../../../bundles/cert-manager

- clusterversion.yaml

patches:
- 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)

Red Hat

Addopters

Lessons learned, lessons shared

OPERATE FIRST>>

NERC

OS-C

MOC ALLIANCE

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_