PUZZLE ITC
changing IT for the better

Im Maschinenraum von Kubernetes

Operator-Bau mit Ansible

Daniel Kobras

2025-03-22 • Chemnitzer Linux-Tage

PUZZLE ITC

changing IT for the better

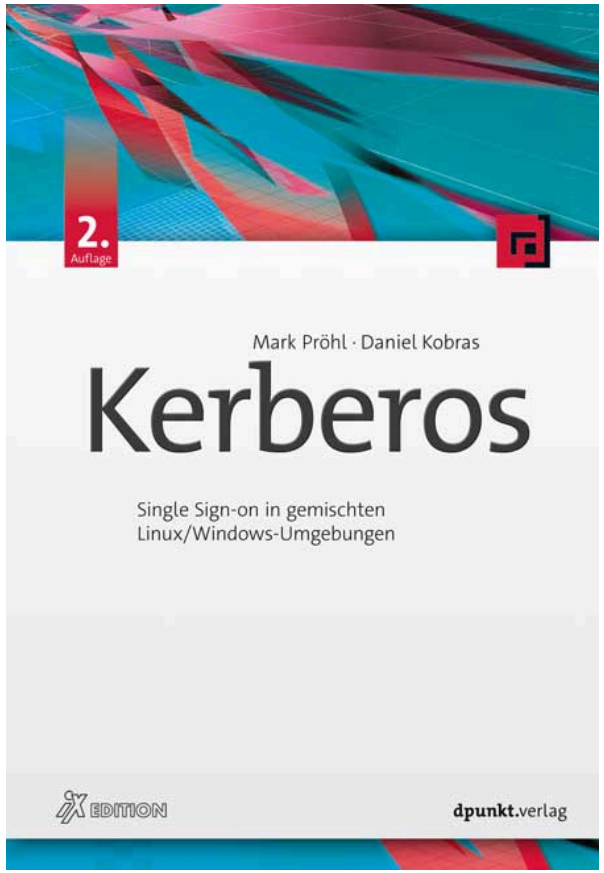
- IT-Dienstleister in CH (Bern, Zürich, Basel) und DE (Tübingen)
- 25 Jahre, 150 Members
- Schwerpunkte:
  - Open-Source-Technologien
  - Applikationsentwicklung
  - Platform Engineering
  - Linux System Engineering
  - Mobility
- https://www.puzzle.ch/

Über mich

Daniel Kobras

Principal Architect (Puzzle ITC)
Kubernetes/Container, Digital Identity, Kerberos-Buch

kobras@puzzle-itc.de



Agenda

Operatoren in Kubernetes
Operatoren im Eigenbau
Beispiele für Ansible-Operatoren

Operatoren in Kubernetes

Was sind Operatoren?

> *Erweiterung der Kubernetes-API um zusätzliche Funktionen*

Bestandteile:

- Controller: Überwacht den Zustand der durch den Operator verwalteten Ressourcen und führt notwendige Änderungen durch, um einen gewünschten Zielzustand zu erreichen (Reconcile Loop).
- Custom Resource Definitions (CRDs): Legt fest, wie der gewünschte Zielzustand syntaktisch beschrieben wird.

Wie benutzt man Operatoren?

Cluster-Admins machen Operator im Cluster verfügbar:

- spielen CRDs ein;
- starten Controller-Pods als Workload im Cluster;
- berechtigen User per Kubernetes-RBAC, Operator zu nutzen.

Berechtigte User nutzen Operator, indem sie der CRD entsprechende Custom Resources (CR) über Kubernetes-API einspielen.

Warum Operatoren?

Native Integration in Kubernetes und dessen Konzepte
Deklarative Konfiguration des gewünschten Zielzustands
Keine externe Automatisierungsinfrastruktur nötig
Permanente Überwachung verwalteter Ressourcen einfach möglich

Warum eigene Operatoren?

Als Alternative zu externer Automatisierung (z.B. über CI/CD-Pipelines, AAP/AWX, ...)
Als Alternative zu Policy Engines (Beschränkung über CR und RBAC statt Policy)
Als Alternative zu ad-hoc-Automatisierung im Cluster (Skript-Jobs, Init-Container)

> *Als Bestandteil einer deklarativen, GitOps-basierten Cluster-Administration*

Operatoren im Eigenbau

Operator-Werkzeuge

Subprojekte in Kubernetes SIG API Machinery für Operator-Entwicklung in Go, z.B.

- controller-runtime/controller-tools (Bibliotheken für Controller-Entwicklung)
- kubebuilder (Scaffolding, SDK)

Operator Framework stellt darauf aufbauend weitere Funktionen für Entwicklung bereit

- Operator Lifecycle Manager für Verwaltung
- OperatorHub für Bereitstellung und Verteilung
- Operator SDK erweitert kubebuilder-Funktionen jenseits Go

Funktionsumfang Operator SDK

Native Entwicklung in Go oder

Plugins für Operator-Implementierung

- in Java
- aus Helm-Charts
- über Ansible-Roles/Playbooks

Passende Container-Images als Basis für eigene Controller

Operatoren mit Ansible

Flexibler als Helm, umfassendere Logik implementierbar

Einfacher als Go/Java, vergleichsweise flache Lernkurve

System Engineers ggf. vertrauter mit Ansible als mit Programmiersprachen

Automatisierung externer Komponenten über bereits bestehende Module (z.B. für Hypervisor, Netzwerkkomponenten, IdM)

Beispiele

Demo 1: Minimal-Operator

- Erstellt ConfigMap aus allen verfügbaren Ansible-Variablen
- Benötigt Modul `k8s`
- Benötigt zusätzliche Berechtigung zum Zugriff auf `ConfigMap`-Ressourcen
- Konfiguration über CR `AnsibleVars`
- Wie werden Einträge in `spec` der CR an Ansible übergeben?
- Was passiert beim Löschen der CR?

Demo 1: Minimal-Operator

Vorbereitung:

```
% operator-sdk init --domain puzzle.ch --plugins ansible
% operator-sdk create api --groups samples --kind AnsibleVars --version v1alpha1 --generate-role
```

Demo 1: Minimal-Operator

Hilfreiche `Makefile`-Anpassungen für Entwicklung mit `kind` (optional)

```
IMG ?= $(IMAGE_TAG_BASE):$(VERSION)
KIND_CLUSTER_CONTEXT ?= kind-cluster

.PHONY: kind-push
kind-push:
	kind load docker-image -n $(KIND_CLUSTER_CONTEXT) ${IMG}
```

Demo 1: Minimal-Operator

`tasks/main.yml`

```yaml
# tasks file for AnsibleVars
- name: Create ConfigMap from all current Ansible variables
  k8s:
    definition:
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: '{{ ansible_operator_meta.name }}'
        namespace: '{{ ansible_operator_meta.namespace }}'
      data:
        vars: |-
          '{{ vars | to_nice_yaml}}'
```

Demo 1: Minimal-Operator

Erzeugen und einspielen:

```
make docker-build docker-push # alternativ kind-push (s.o.)
make deploy
```

Demo 1: Minimal-Operator

Test mit

```
kubectl apply -f config/samples/samples_v1alpha1_ansiblevars.yaml
```

Demo 2: DbAccount-Operator

- Ressourcen abfragen (Modul `k8s_info`)
- Status verwalten (Modul `k8s_status`)
- Finalizer (über `watches.yml`)
- Syntaxprüfung über CRD (`properties`, `required`, `x-kubernetes-validations`)
- Ausgabe steuern über CRD (`additionalPrinterColumns`)

Zusammenfassung

Fazit

Operatoren erweitern Kubernetes um zusätzliche Funktionen
Operator SDK vereinfacht Entwicklung eigener Operatoren
Verknüpfung mit Ansible bietet niedrige Einstiegshürde trotz hohem Funktionsumfang

> *Automatisierung administrativer Prozesse gepaart mit cloud-nativen Konzepten (GitOps, Kubernetes-RBAC)*

Ausblick

Tests und Validierung
Catalog-Verwaltung
Upgrades/Lifecycle Management

Referenzen

Online

- Operator SDK
- Kubebuilder-Dokumentation

Print

- Kubernetes Patterns
- The Kubernetes Operator Framework Book
- Operation einfach, iX 3/2022, S. 122f

Vielen Dank!

Daniel Kobras kobras@puzzle-itc.de