



# Operator in a Nutshell

SVA

Norris Sam Osarenkhoe, DevOps Architect

# Die Kubernetes Journey

Gut Ding will Weile haben

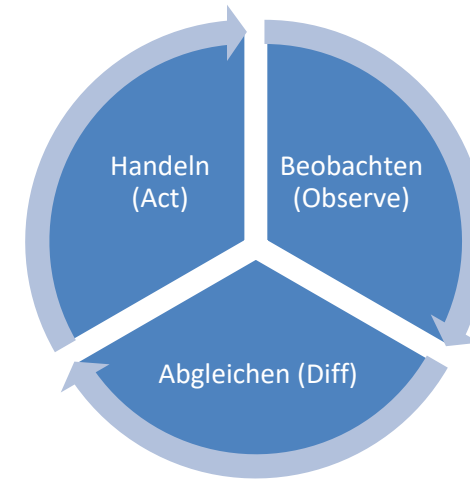
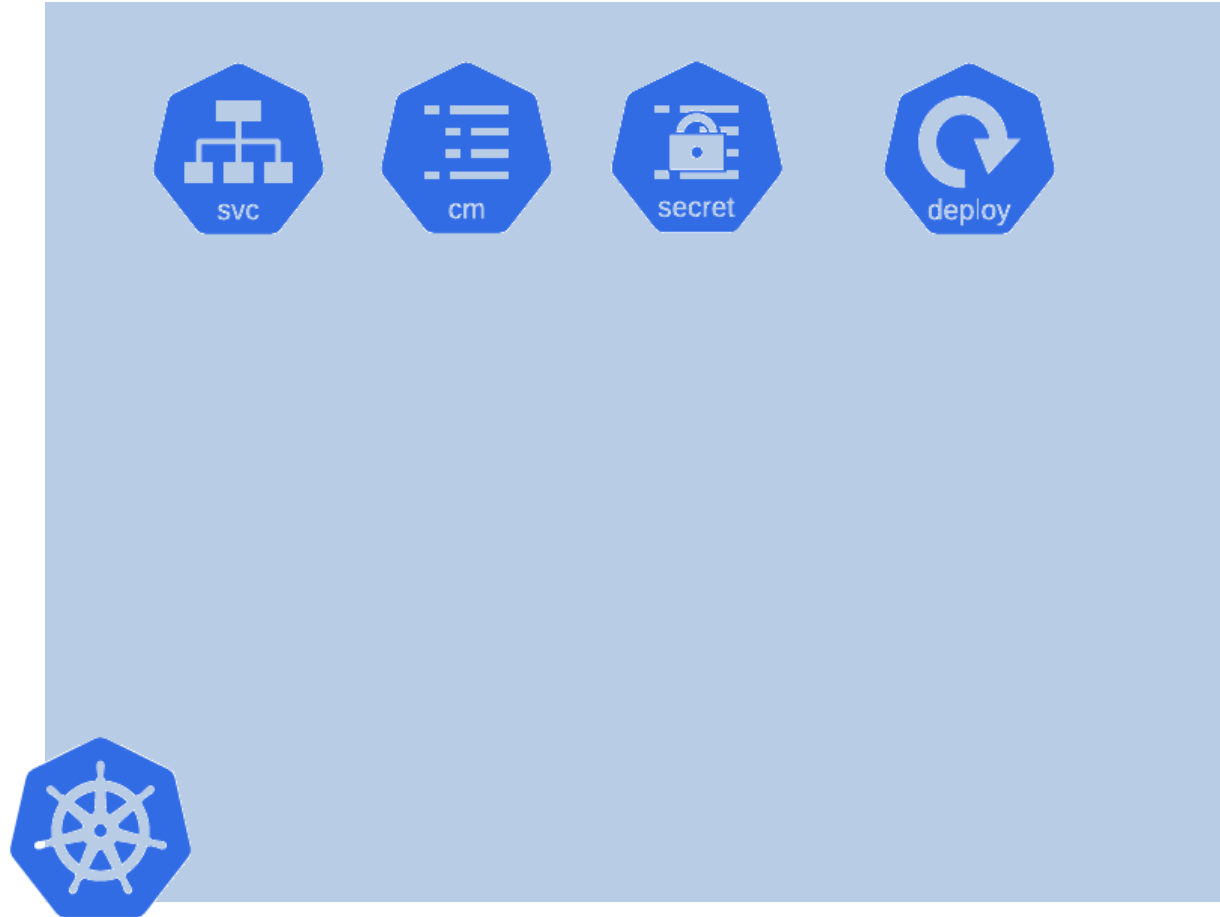
# / Phase 1: Auf geht's!



meine-api-v1



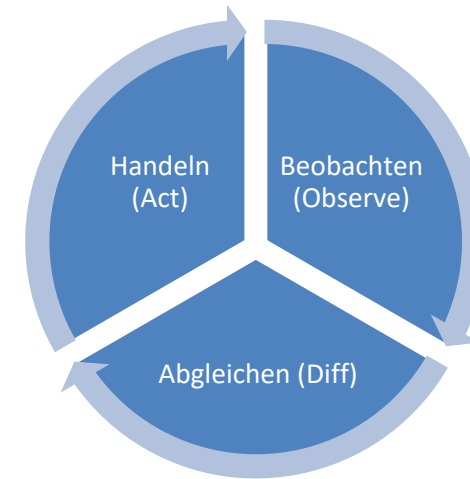
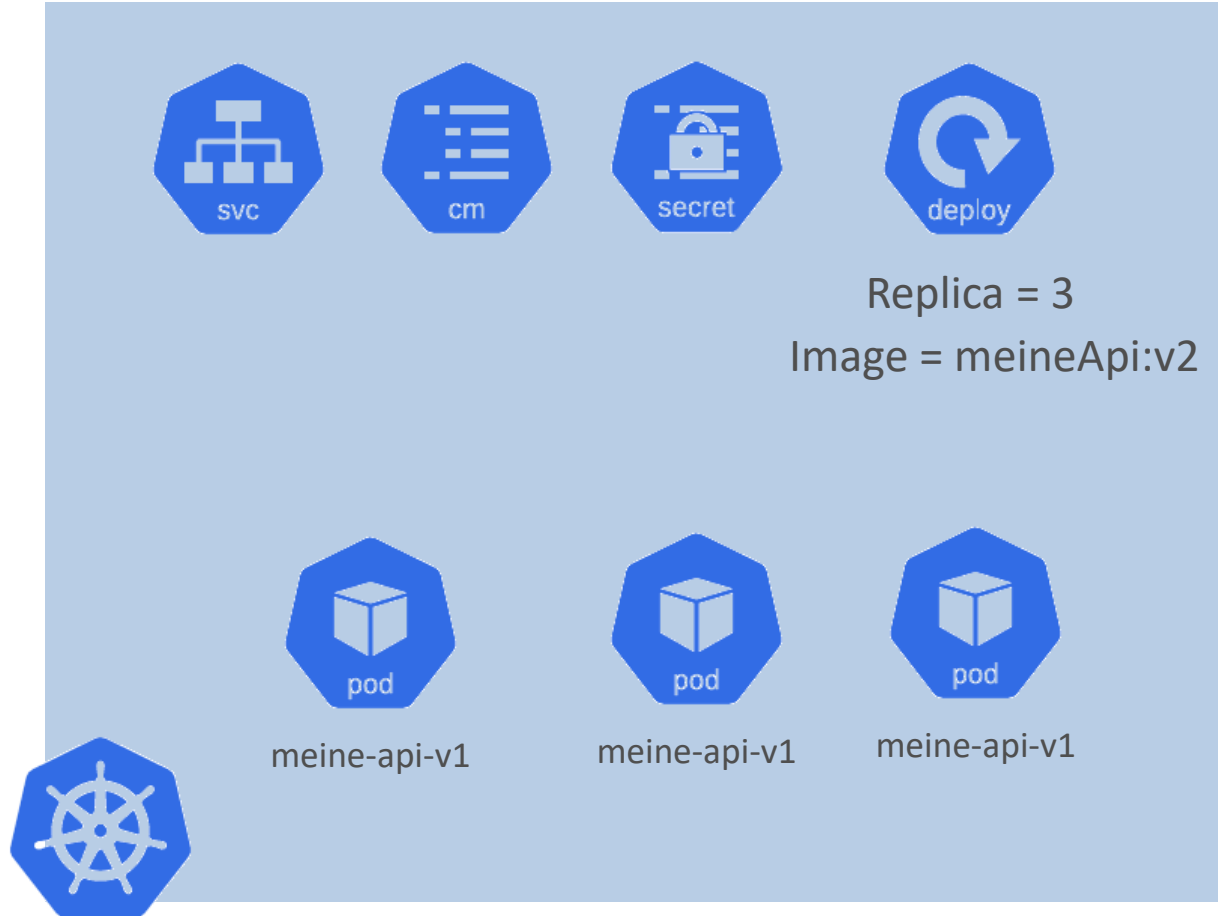
# / Phase 1: Auf geht's!



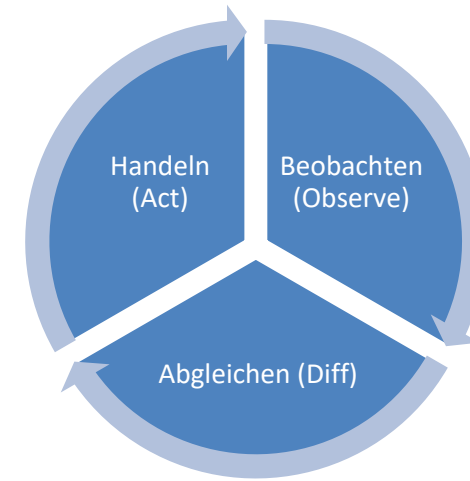
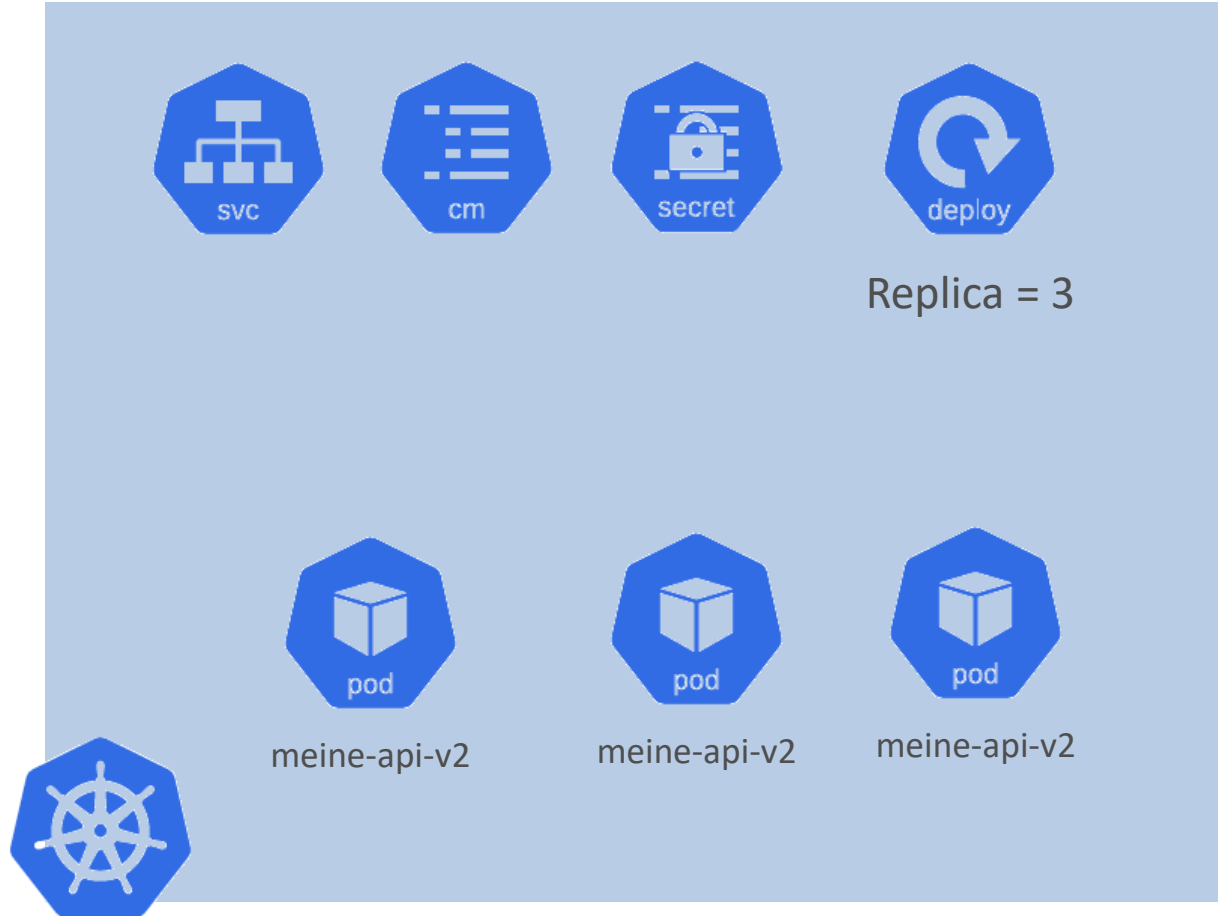
meine-api-v1



# / Phase 1: Auf geht's!



# / Phase 1: Auf geht's!



# / Phase 1: Auf geht's!

- Kennenlernen der Basis Ressourcen Typen
  - Pod, Deployment, Service, ReplicaSet, Secret, ConfigMap
- Erste Versuche Kubernetes zu bändigen erfolgen
  - `kubectl` steht uns als treuer Verbündeter zur Seite
- Es werden immer mehr YAML Dateien
  - Helm wird entdeckt

Fazit Phase 1:

Kubernetes hat uns überzeugt. Super Sache!



## / Phase 2: Don't Panic!

- Der Erfolg hat sich rumgesprochen:
  - Es sollen nun auch komplexe Anwendungen wie Datenbanken in das Cluster.
- Weitere Ressourcen Typen und Konzepte werden entdeckt:
  - [StatefulSets](#), [PersistentVolumes](#), [Jobs](#)
- Es werden immer mehr YAML Dateien
  - [Helm](#) Charts werden nun selber riesig. Es werden nun aufwendige Runbooks geschrieben, um Fehler bei der Konfiguration zu vermeiden und gängige Szenarien abzudecken (z.B. Backups).

Fazit Phase 2:

Ganz schön aufwendig. Wie soll das mit 3 Leuten skalieren?



# / Kubernetes Resource Basics



k8s Resource

- Resource Definition
- Resource Controller



k8s API

# Phase 3: Neue Wege.

# / Custom Resource Definition


- Beschreibt und Registriert die CRD gegenüber der Kubernetes API
- CRD wird somit wie andere Ressourcen Teil der Control Loop
  - D.h. Events zur CR werden nun durch Kubernetes veröffentlicht und man kann diesen Events lauschen.
  - Ermöglicht uns die standardmäßige Control Loop, um unsere benutzerdefinierte Logik zu ergänzen.

```
1  apiVersion: apiextensions.k8s.io/v1beta1
2  kind: CustomResourceDefinition
3  metadata:
4    name: etcdclusters.etcd.database.coreos.com
5  spec:
6    group: etcd.database.coreos.com
7    names:
8      kind: EtcdCluster
9      listKind: EtcdClusterList
10     plural: etcdclusters
11     shortNames:
12       - etcdclus
13       - etcd
14     singular: etcdcluster
15     scope: Namespaced
16     versions:
17       - name: v1beta2
18         served: true
19         storage: true
20
```



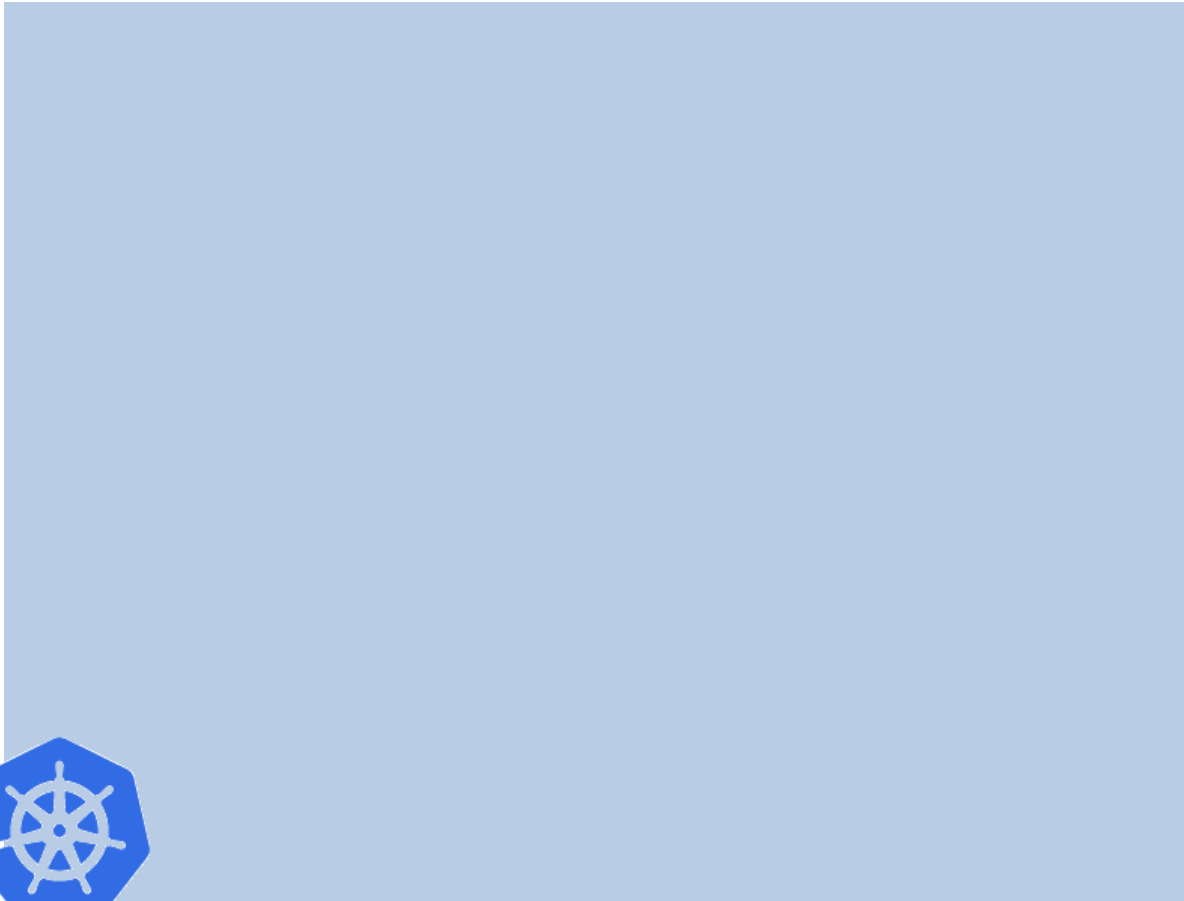
# / Custom Resource Controller

- Unser Programm, welches die Kubernetes Control Loop um unsere benutzerdefinierte Control Loop ergänzt.

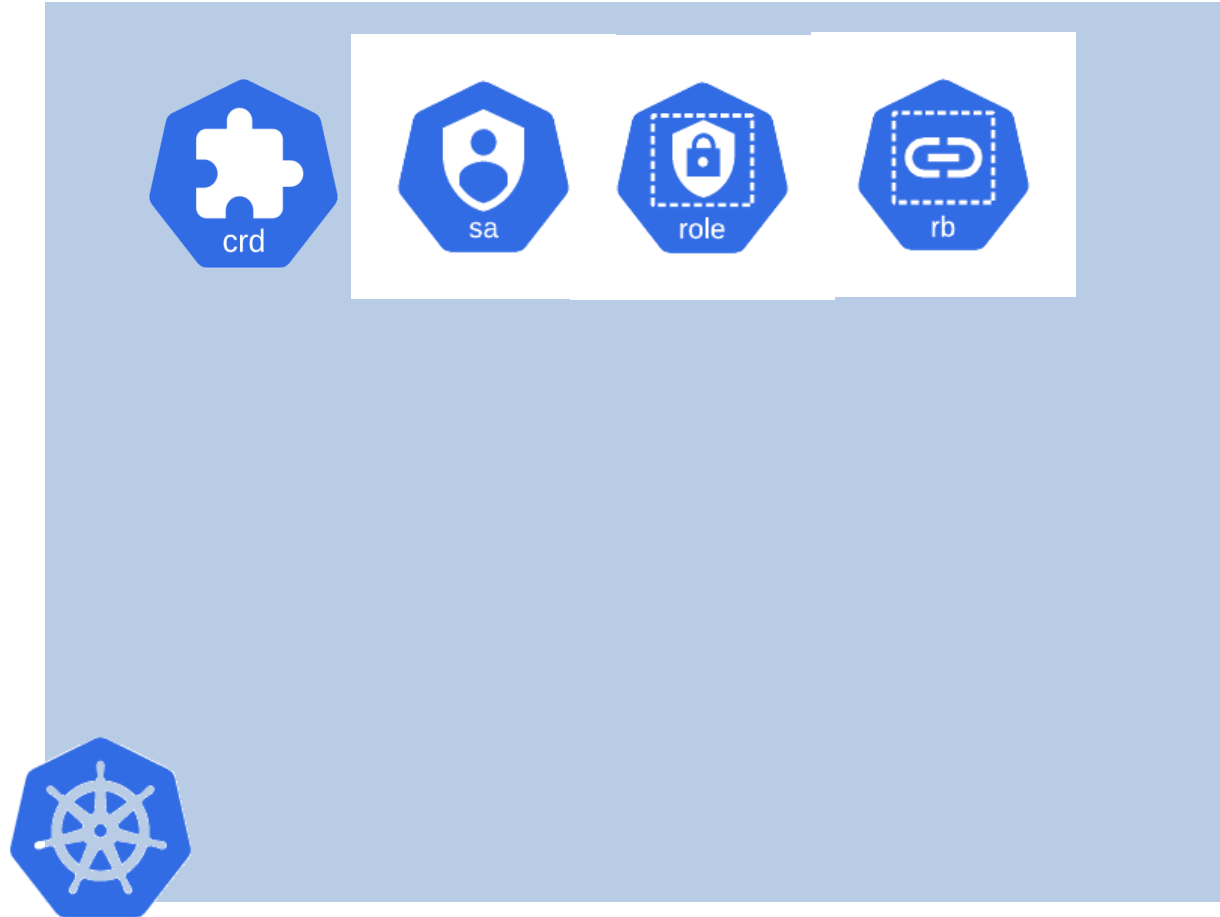


```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: etcd-operator
5  spec:
6    selector:
7      matchLabels:
8        app: etcd-operator
9    replicas: 1
10   template:
11     metadata:
12       labels:
13         app: etcd-operator
14     spec:
15       containers:
16         - name: etcd-operator
17           image: quay.io/coreos/etcd-operator:v0.9.4
18           command:
19             - etcd-operator
20             - --create-crd=false
21           env:
22             - name: MY_POD_NAMESPACE
23               valueFrom:
24                 fieldRef:
25                   fieldPath: metadata.namespace
26             - name: MY_POD_NAME
27               valueFrom:
28                 fieldRef:
29                   fieldPath: metadata.name
30           imagePullPolicy: IfNotPresent
31       serviceAccountName: etcd-operator-sa
```

## / Phase 3: Neue Wege.



## / Phase 3: Neue Wege.



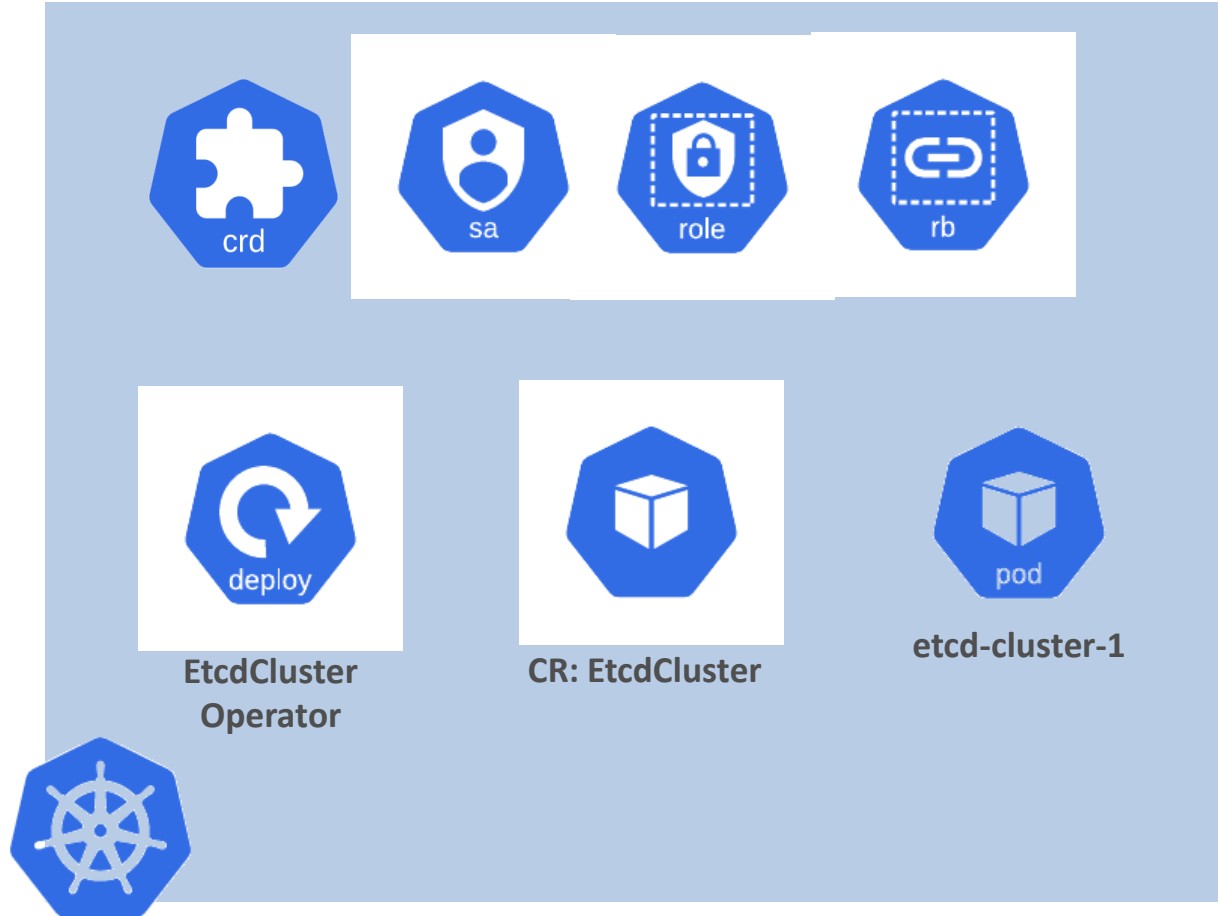
CR: EtcdCluster



EtcdCluster  
Operator



## / Phase 3: Neue Wege.



# Demo





Take-Away



- Operator erweitern die standardmäßigen Kubernetes Ressourcen.
- Mit Ihnen können zustandsbehaftete, individuelle und komplexe Softwarelösungen in einem Kubernetes Cluster bereitgestellt werden.
- Operator erlauben das spezifische und über Jahre gebildete Wissen des Betriebs solcher komplexen Lösungen als ausführbaren Code zu definieren.
- Dadurch werden wiederkehrende, monotone Konfigurations- und Wartungstätigkeiten automatisiert.
- Treten wiederkehrende Probleme auf, wird nicht an der Software selber gearbeitet, sondern ein Patch für den Operator veröffentlicht.
- Dies erlaubt eine Entkopplung zwischen Fehlerbehebung und Bereitstellung eines Patches. Die Bereitstellung kann so simpel sein wie ein „kubectl apply“. Dies erlaubt Skalierung.



# / Weiterführende Links

- MySQL Operator: <https://github.com/oracle/mysql-operator>
- Hands-On Kurs 1: <https://learn.openshift.com/ansibleop/ansible-operator-overview>
- Weiterführende Kurse: <https://developers.redhat.com/courses/openshift-operators>
- E-Book O'reilly: <https://developers.redhat.com/books/kubernetes-operators>
- Marketplace mit Openshift zertifizierten Operator:
  - <https://marketplace.redhat.com/en-us>
- Operator-SDK:
  - <https://sdk.operatorframework.io/>
- Ansible-Kubernetes Plugin:
  - [https://docs.ansible.com/ansible/latest/collections/community/kubernetes/k8s\\_module.html](https://docs.ansible.com/ansible/latest/collections/community/kubernetes/k8s_module.html)
- Marketplace mit diversen „Out-Of-The-Box“ Operator:
  - <https://operatorhub.io/>