Aufbau und Dokumentation von ApplicationSet's für ArgoCD zum deployen von DataBase Instanzen

grobe Gedanken zum generellen Aufbau im Cluster und in der ArgoCD

1. in ArgoCD ein Projekt pro DB Type (mariadb, postgres...)
2. im Git wird ein Repo pro DB Type angelegt
    1. darin werden Ordner für produktion und referenz abgelegt in welche dann die jeweiligen DB Instanzen als Ordner mit darin enthaltenen Manifestfiles angelegt werden
    2. dadurch wird pro DB Instanz und der damit einhergehenden Deklaration eine Anwendung in ArgoCD angelegt
3. es soll pro DB Instanz jeweils eine Anwendung und damit verbunden ein Namespace angelegt werden
    1. beides wird jeweils den selben Namen bekommen

Verwendung von ApplicationSet, um Anwendungen dynamisch zu erzeugen

- Erstellen einer ApplicationSet-Ressource, die Anwendungen automatisch erzeugt, sobald ein neuer Ordner oder eine neue Anwendung im Git-Repository hinzugefügt wird.
- Ein Beispiel für ein ApplicationSet, das auf ein Git-Repository verweist und für jedes Verzeichnis darin eine Anwendung erstellt:

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: postgres-referenz-fachverfahren
  namespace: argocd
spec:
  generators:
    - git:
        repoURL: https://git.cooltux.net/dbaas/postgres.git
        revision: HEAD
        directories:
          - path: "referenz/*" # Sucht nach allen Verzeichnissen in
"referenz/*"
  template:
    metadata:
      name: "{{path.basename}}" # Setzt den Namen basierend auf dem
Verzeichnisnamen
    spec:
```

```
      project: dbaas
      source:
        repoURL: https://git.cooltux.net/dbaas/postgres.git
        targetRevision: HEAD
        path: "{{path}}" # Nutzt den spezifischen Verzeichnispfad (z.B.
apps/nginx)
      destination:
        server: https://k8s-control1:6443
        namespace: "{{path.basename}}" # Namespace wird basierend auf dem
Verzeichnisnamen erstellt
      syncPolicy:
        automated:
          prune: true # Entfernt nicht mehr genutzte Ressourcen
          selfHeal: true # Automatische Wiederherstellung, falls der
Clusterzustand abweicht
        syncOptions:
          - CreateNamespace=true # Hier wird sichergestellt, dass der
Namespace automatisch angelegt wird
          - ServerSideApply=true # Aktiviert Server-Side Apply
```

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: postgres-produktion-fachverfahren
  namespace: argocd
spec:
  generators:
    - git:
        repoURL: https://git.cooltux.net/dbaas/postgres.git
        revision: HEAD
        directories:
          - path: "produktion/*" # Sucht nach allen Verzeichnissen in
"produktion/"
  template:
    metadata:
      name: "{{path.basename}}" # Setzt den Namen basierend auf dem
Verzeichnisnamen
    spec:
      project: dbaas
      source:
        repoURL: https://git.cooltux.net/dbaas/postgres.git
        targetRevision: HEAD
        path: "{{path}}" # Nutzt den spezifischen Verzeichnispfad (z.B.
produktion/instanz/10010067)
      destination:
        server: https://k8s-control1:6443
        namespace: "{{path.basename}}" # Namespace wird basierend auf dem
Verzeichnisnamen erstellt
      syncPolicy:
        automated:
```

```
        prune: true # Entfernt nicht mehr genutzte Ressourcen
        selfHeal: true # Automatische Wiederherstellung, falls der
Clusterzustand abweicht
      syncOptions:
        - CreateNamespace=true # Hier wird sichergestellt, dass der
Namespace automatisch angelegt wird
        - ServerSideApply=true # Aktiviert Server-Side Apply
```

Erklärung:

- Der Generator (generators: git) durchsucht das angegebene Git-Repository nach Verzeichnissen unter dem Pfad production/*.
- Jedes Verzeichnis wird als neue Anwendung erkannt und automatisch durch die template-Spezifikation ausgerollt.
- syncPolicy.automated stellt sicher, dass alle Änderungen im Git automatisch synchronisiert werden, ohne manuellen Eingriff.

GitOps-Verzeichnisstruktur festlegen

- Legen Sie die Struktur des Git-Repositories so an, dass jedes Verzeichnis unterhalb des Verzeichnis des jeweiligen DB Typennamen eine separate Anwendung darstellt. Zum Beispiel könnte Ihre Verzeichnisstruktur folgendermaßen aussehen:

```
dbaas-postgres/
  ├── referenz/
      ├── instanz1/
          ├── Cluster.yml
      ├── instanz2/
          ├── Cluster.yml
  ├── production/
      ├── instanz1/
          ├── MariaDB.yml
      ├── instanz2/
          ├── MariaDB.yml

dbaas-mariadb/
  ├── referenz/
      ├── instanz3/
          ├── Cluster.yml
      ├── instanz4/
          ├── Cluster.yml
  ├── production/
      ├── instanz3/
          ├── MariaDB.yml
      ├── instanz4/
          ├── MariaDB.yml
```

- Automatische Synchronisierung und Überwachung
  - Argo CD wird automatisch erkennen, wenn neue Verzeichnisse im postgres/ Pfad hinzugefügt oder verändert werden, und die entsprechenden Anwendungen ausrollen.
  - Durch das Setzen von automated in der syncPolicy werden Änderungen ohne manuelles Zutun synchronisiert.

Zusammenfassung

Mit diesem Setup erreichen Sie Folgendes:

- Automatische Erkennung und Bereitstellung: Neue Anwendungen werden automatisch durch das ApplicationSet erkannt und ausgerollt, wenn sie im Git abgelegt werden.
- Keine manuelle Interaktion: Es wird weder die Argo CD CLI noch die GUI benötigt, da alle Konfigurationen durch die Git-basierten Manifeste durchgeführt werden.
- Vollständige Automatisierung: Jede Änderung im Git-Repository wird automatisch ausgerollt, solange die Anwendung im festgelegten Verzeichnis liegt.

Dies ist eine sehr effektive Möglichkeit, um den vollen Vorteil des GitOps-Ansatzes zu nutzen, bei dem das Git-Repository die einzige Quelle der Wahrheit ist und die gesamte Deployment-Automatisierung darauf basiert.