Parametrisierte Kubernetes Deployments mit kustomize

In Kubernetes wird mithilfe von YAML-Konfigurationen festgelegt, auf welche Weise Applikationen in der Infrastruktur bereitgestellt werden. Aber was, wenn wir unsere Applikation für mehrere verschiedene Umgebungen konfigurieren möchten? Für jede Umgebung eine einzelne YAML-Konfiguration zu erstellen ist oft mühselige Arbeit. Zum Glück existiert mit Kustomize ein Tool, welches genau diesen Arbeitsschritt automatisieren kann. Wie funktioniert Kustomize? Bevor wir uns mit dieser Frage beschäftigen, schauen wir uns erst einmal an einem Beispiel an, wie eine manuelle Konfiguration für verschiedene Umgebungen aussieht.

In unserem Szenario läuft auf unserer Dev-Umgebung bisher ein Service, welcher über das HTTP-Protokoll auf Port 80 erreichbar ist. Dieser Service soll nun auch auf der Testumgebung erreichbar sein, dort allerdings aus Sicherheitsgründen über HTTPS auf dem Port 443. Um dies umzusetzen, muss für jede Umgebung eine separate Konfiguration angelegt werden:

Konfiguration für die Dev-Umgebung:

```yaml
kind: Service
apiVersion: v1
metadata:
  name: env-anzeige-frontend-https
spec:
  type: LoadBalancer
  selector:
    app: env-anzeige-frontend
  ports:
  - name: http
    port: 80
```

Konfiguration für die Testumgebung mit geänderten Porteinstellungen:

```yaml
kind: Service
apiVersion: v1
metadata:
  name: env-anzeige-frontend-https
spec:
  type: LoadBalancer
  selector:
    app: env-anzeige-frontend
  ports:
  - name: https
    port: 443
```

Um den Service für die Testumgebung zu konfigurieren, wurde die Dev-Konfig kopiert und nur der relevante Datensatz – die Porteinstellungen – geändert. Und genau an diesem Punkt gehen die

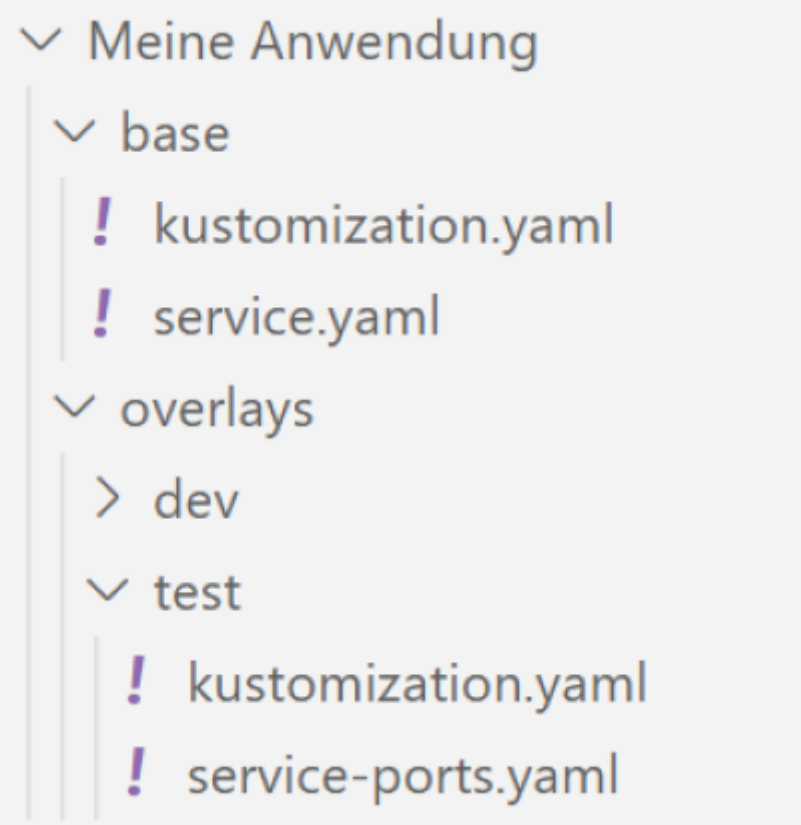
Probleme los. Die YAMLs ständig manuell zu kopieren und anzupassen ist auf Dauer sehr zeitaufwändig. Außerdem entstehen durch das ständige „Hin-und-her-kopieren“ verschiedene Konfigurationen, welche größtenteils identisch sind. Dadurch ist im Nachhinein schnell nicht mehr klar, was denn nun der aktuelle Master-Stand ist.

Kustomize kann diese Probleme lösen, indem es die Anpassung bestehender Ressourcen vereinfacht und automatisiert. Dabei arbeitet Kustomize nach einem einfachen Grundkonzept: Es wird nur eine einzige Basiskonfiguration für die Bereitstellung der Anwendung angelegt. Für die einzelnen Umgebungen werden nur die an der Basis durchzuführenden Patches, die sogenannten Overlays, abgelegt.

Schauen wir uns einmal an, wie wir unser Beispiel mit den Porteinstellungen in Kustomize umsetzen können. Als Erstes muss eine *kustomization.yaml* angelegt werden, welche alle benötigten Ressourcen und die zu verwendenden Patches auflistet.

```
#base/kustomization.yaml
resources:
- service.yaml
```

```
#overlays/test/kustomization.yaml
bases:
- ../../base
patches:
- service-ports.yaml
```



Durch Patches können einzelne YAML-Attribute einer Konfiguration hinzugefügt oder überschrieben werden. Hierzu wird eine neue YAML-Datei angelegt, welche den Namen der zu patchenden Ressource und alle zu ändernden Attribute beinhaltet.

```
#overlays/test/service-ports.yaml
kind: Service
apiVersion: v1
metadata:
  #Name der zu patchenden Ressource
  name: env-anzeige-frontend-https
spec:
  ports: #Die Porteinstellungen werden überschrieben
  - name: https
    port: 443
```

Durch die Erstellung von Patch-Dateien können einzelne YAML-Attribute einer Konfiguration hinzugefügt oder überschrieben werden. Um anspruchsvollere Veränderungen an den Dateien durchzuführen, bietet Kustomize zusätzlich das Verändern von Ressourcen unter Verwendung des JSON6902 Standards an. Dieser beinhaltet sechs weitere Operatoren zur Veränderung von Ressourcen und ist damit umfangreicher als der Standard-Patch.

JSON6902 Operator	**Beschreibung**
Add	fügt ein Attribut am angegebenen Pfad hinzu
Remove	entfernt ein Attribut
Replace	überschreibt das Attribut am angegebenen Pfad mit einem neuen Wert
Move	verschiebt ein Attribut an einen anderen Pfad
Copy	kopiert ein Attribut und fügt es an einem anderen Pfad wieder ein
Test	überprüft, ob ein bestimmtes Attribut den angegebenen Wert besitzt

Im Folgenden wird der Service-Patch mit der JSON-Methode durchgeführt:

```
#kustomization.yaml
patchesJson6902:
- target:
    version: v1
    kind: Service
    name: env-anzeige-frontend-https
  path: service-patch.yaml
```

```
#service-patch.yaml
- op: replace
  path: /spec/ports
  value:
  - name: https
    port: 443
```

Die herkömmlichen Patches haben einen kleineren Funktionsumfang als der JSON-Patch, dadurch sind sie aber auch lesbarer. Außerdem reichen die Operationen hinzufügen und überschreiben für die meisten Anwendungsfälle völlig aus. Ein JSON-Patch sollte daher nur verwendet werden, wenn die zusätzlichen Operatoren (Attribut entfernen/kopieren/verschieben/testen) benötigt werden.

Nachdem wir unsere Patches in Kustomize deklariert haben, können wir nun die einsetzbaren YAML-Konfigurationen erstellen. Kustomize ist fest in der aktuellen Version von kubectl implementiert. Mit dem Befehl **kubectl kustomize** kann die gepatchte YAML Konfigurationen ausgeben lassen.

```
> kubectl kustomize ./overlays/test/
```

```
apiVersion: v1
kind: Service
metadata:
  name: env-anzeige-frontend-https
spec:
  ports:
  - name: https
    nodePort: 30951
    port: 443
  selector:
    app: env-anzeige-frontend
  type: LoadBalancer
```

Alternativ können die Ressourcen mit dem Befehl **kubectl apply -k** ohne Zwischenschritt direkt in das Cluster geladen werden.

```
> kubectl apply -k ./overlays/test/
service/env-anzeige-frontend-https created
```

Fazit

Kustomize vereinfacht den Konfigurationsprozess für verschiedene Umgebungen erheblich. Weitere Vorteile von Kustomize sind die feste Implementierung in kubectl. Dadurch ist keine weitere Konfiguration oder Installation notwendig.