

Übung: kubectl-CLI und Basis Ressourcen (YAML Variante)

Das `kubectl` -Kommando stellt, eine der Schaltzentralen des K8s Clusters zur Administration der Ressourcen dar.

Die `YAML` Beschreiben die Ressourcen und Vereinfachen so die Verwendung des `kubectl` Kommandos.

In dieser Übung verwenden wir das `kubectl` -Kommando mit `YAML` Dateien zur Erstellen eines Pods und Services.


Das passiert in einer eigenen Namespace um die Resultate gezielt Darstellen zu können:

```
In [1]: ! kubectl create namespace yaml  
namespace/yaml created
```

Für den Pod haben wir, die Ausgabe von `kubectl get pod apache -o yaml` genommen und eine abgespeckte Variante als `YAML` Datei erstellt:

```
In [2]: ! cat 09-2-YAML/apache-pod.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
  labels:  
    app.kubernetes.io/name: apache  
  name: apache  
  namespace: yaml  
spec:  
  containers:  
  - image: httpd  
    name: apache
```

Die Erklärung der Einträge liefert uns

In [3]:  ! kubectl explain pod

```
KIND:      Pod
VERSION:   v1
```

DESCRIPTION:

Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.

FIELDS:

```
  apiVersion  <string>
    APIVersion defines the versioned schema of this representation of an
    object. Servers should convert recognized schemas to the latest internal
    value, and may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#resources (https://git.k8s.io/community/contributors/devel/api-conventions.md#resources)

  kind <string>
    Kind is a string value representing the REST resource this object
    represents. Servers may infer this from the endpoint the client submits
    requests to. Cannot be updated. In CamelCase. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds (https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds)

  metadata    <Object>
    Standard object's metadata. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata (https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata)

  spec <Object>
    Specification of the desired behavior of the pod. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status (https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status)

  status      <Object>
    Most recently observed status of the pod. This data may not be up to date.
    Populated by the system. Read-only. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status (https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status)
```

Mit diesen Informationen können wir anhand der YAML Datei den Pod erzeugen

```
In [4]: ! kubectl create -f 09-2-YAML/apache-pod.yaml  
pod/apache created
```

Zur Kontrolle geben wir den Pod aus:

```
In [5]: ! kubectl get pods,services --namespace yaml  
  
NAME          READY   STATUS             RESTARTS   AGE  
pod/apache    0/1     ContainerCreating   0          1s
```

Für den Service verfahren wir gleich. Ausgabe von `kubectl get service apache -o yaml` Analysieren und daraus eine YAML Datei erstellen.

Durch den Selector

```
selector:  
  app.kubernetes.io/name: apache
```

werden Pod und Service verbunden.

In [6]: `! cat 09-2-YAML/apache-service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/name: apache
  name: apache
  namespace: yaml
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app.kubernetes.io/name: apache
  type: LoadBalancer
```

In [7]: `! kubectl create -f 09-2-YAML/apache-service.yaml`

service/apache created

Wir sollten jetzt einen Pod und einen Service apache haben

In [8]: `! kubectl get pods,service apache --namespace yaml`

NAME	READY	STATUS	RESTARTS	AGE
pod/apache	0/1	ContainerCreating	0	2s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/apache	LoadBalancer	10.98.202.241	<pending>	80:32354/TCP	0s

Da wir keinen LoadBalancer haben müssen wir mit einem kleinen Shellscript selber die IP des Clusters und der gemappte Port als URL aufbereiten.

Diese Shellscript ist im Script `startsvc` hinterlegt.

http://192.168.178.200:32354 (http://192.168.178.200:32354)

Zum Aufräumen genügt es den Namespace zu löschen

```
namespace "yaml" deleted
```