Lab 3: Ein Docker Image deployen

In diesem Lab werden wir das erste “pre-built” Docker Image deployen und die OpenShift-Konzepte Pod, Service, DeploymentConfig und ImageStream noch etwas genauer anschauen.

Aufgabe: LAB 3.1

Nachdem wir im Lab 2 den Source-to-Image Workflow verwendet haben, um eine Applikation auf OpenShift zu deployen, wenden wir uns nun dem Deployen eines pre-built Docker Images von Docker Hub oder einer anderen Docker-Registry widmen.

Weiterführende Dokumentation

Als ersten Schritt, erstellen wir dazu ein neues Projekt. Ein Projekt ist wie schon in der Einführung beschrieben, eine Gruppierung von Ressourcen (Container, Docker Images, Pods, Services, Routen, Konfiguration, Quotas, Limiten und weiteres). Für das Projekt berechtigte User können diese Ressourcen verwalten. **Achtung: Innerhalb eines OpenShift v3 Clusters muss der Name eines Projektes eindeutig sein.**

Wir erstellen nun daher ein neues Projekt mit dem Namen [DEINNAME]-dockerimage:

```
# oc new-project [DEINNAME]-dockerimage
```

oc new-project wechselt automatisch in das eben neu angelegte Projekt. Mit dem **oc get** Command können Ressourcen von einem bestimmten Typ angezeigt werden.

Man verwendet

```
# oc get project
```

um alle Projekte anzuzeigen, auf die man berechtigt ist.

Sobald das neue Projekt erstellt wurde, wird dies auch gleich als aktuell aktives Projekt gewählt. Wir können nun also nun in OpenShift mit dem folgenden Befehl das Docker Image so direkt in das richtige Projekt deployen:

```
# oc new-app appuio/example-spring-boot
```

```
--> Found Docker image d790313 (3 weeks old) from Docker Hub for
"appuio/example-spring-boot"

    APPUiO Spring Boot App
    ----------------------
```

```
    Example Spring Boot App

    Tags: builder, springboot

    * An image stream will be created as "example-spring-boot:latest" that
will track this image
    * This image will be deployed in deployment config "example-spring-boot"
    * Port 8080/tcp will be load balanced by service "example-spring-boot"
      * Other containers can access this service through the hostname
"example-spring-boot"

--> Creating resources with label app=example-spring-boot ...
    imagestream "example-spring-boot" created
    deploymentconfig "example-spring-boot" created
    service "example-spring-boot" created
--> Success
    Run 'oc status' to view your app.
```

Für das Lab verwenden wir ein APPUiO-Beispiel (Java Spring Boot Applikation):

- Docker Hub: https://hub.docker.com/r/appuio/example-spring-boot/
- GitHub (Source): https://github.com/appuio/example-spring-boot-helloworld

OpenShift legt die nötigen Ressourcen an, lädt das Docker Image in diesem Fall von Docker Hub herunter und deployt anschliessend den ensprechenden Pod.

Tipp: Verwende `oc status` um dir einen Überblick über das Projekt zu verschaffen.

Oder verwende den `oc get pods` Befehl mit dem **`-w`** `Parameter`, um fortlaufend Änderungen an den Ressourcen des Typs Pod anzuzeigen (abbrechen mit ctrl+c):

```
# oc get pods -w
```

Je nach Internetverbindung oder abhängig davon, ob das Image auf dem OpenShift Node bereits heruntergeladen wurde, kann das eine Weile dauern. Schaue dir dabei doch in der Web Console den aktuellen Status des Deployments an:

1. Logge dich in der Web Console ein
2. Wähle das Projekt [DEINNAME]-dockerimage aus
3. Klicke auf Applications
4. Wähle Pods aus

Wichtig: Um eigene Docker Images für OpenShift zu erstellen, sollte man dabei die folgenden Best Practices befolgen: https://docs.openshift.com/container-platform/3.9/creating_images/guidelines.html

---

Betrachten der soeben erstellten Ressourcen

Als wir `oc new-app appuio/example-spring-boot` vorhin ausführten, hat OpenShift im Hintergrund einige Ressourcen für uns angelegt. Folgende drei Ressourcen werden zentral gebraucht, um ein Docker Image überhaupt korrekt zu deployen:

- Service
- ImageStream
- DeploymentConfig

Service

Die Services dienen innerhalb OpenShift als Abstraktionslayer, Einstiegspunkt und Proxy/Loadbalancer auf die dahinterliegenden Pods. Der Service ermöglicht es, innerhalb OpenShift eine Gruppe von Pods des gleichen Typs zu finden und anzusprechen.

Als Beispiel: Wenn eine Applikationsinstanz unseres Beispiels die Last nicht mehr alleine verarbeiten kann, können wir die Applikation bspw. auf drei Pods hochskalieren. OpenShift mapt diese als Endpoints automatisch zum Service. Sobald die Pods bereit sind, werden Requests automatisch auf alle drei Pods verteilt.

Note: Die Applikation kann aktuell von aussen noch nicht erreicht werden, der Service ist ein OpenShift-internes Konzept. *Erst im nachfolgenden Lab (Lab4) werden wir unsere Applikation öffentlich verfügbar machen.*

Nun schauen wir uns unseren Service mal etwas genauer an:

```
# oc get services
```

```
NAME                  CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
example-spring-boot   172.30.124.20   <none>        8080/TCP   2m
Wie Sie am Output sehen, ist unser Service (example-spring-boot) über eine
IP und Port erreichbar (172.30.124.20:8080) Note: Ihre IP kann
unterschiedlich sein.
```

Note: Die Service IP's bleiben während ihrer Lebensdauer immer gleich.

Mit dem folgenden Befehl können zusätzliche Informationen über den Service ausgelesenen werden:

```
# oc get service example-spring-boot -o json
```

```
{
    "kind": "Service",
```

```
    "apiVersion": "v1",
    "metadata": {
        "name": "example-spring-boot",
        "namespace": "techlab",
        "selfLink": "/api/v1/namespaces/techlab/services/example-spring-
boot",
        "uid": "b32d0197-347e-11e6-a2cd-525400f6ccbc",
        "resourceVersion": "17247237",
        "creationTimestamp": "2018-07-11T 11:29:05Z",
        "labels": {
            "app": "example-spring-boot"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "ports": [
            {
                "name": "8080-tcp",
                "protocol": "TCP",
                "port": 8080,
                "targetPort": 8080
            }
        ],
        "selector": {
            "app": "example-spring-boot",
            "deploymentconfig": "example-spring-boot"
        },
        "portalIP": "172.30.124.20",
        "clusterIP": "172.30.124.20",
        "type": "ClusterIP",
        "sessionAffinity": "None"
    },
    "status": {
        "loadBalancer": {}
    }
}
```

Mit dem entsprechenden Befehl können auch die Details zu einem Pod angezeigt werden:

```
# oc get pod example-spring-boot-3-nwzku -o json
```

Wichtig: Zuerst den pod Namen aus dem Projekt abfragen (`oc get pods`) und im oberen Befehl ersetzen!

Über den selector Bereich im Service wird definiert, welche Pods (labels) als Endpoints dienen. Dazu die entsprechenden Konfigurationen vom Service und Pod zusammen betrachten.

```
Service:
--------
...
"selector": {
    "app": "example-spring-boot",
    "deploymentconfig": "example-spring-boot"
},

...

Pod:
----
...
"labels": {
    "app": "example-spring-boot",
    "deployment": "example-spring-boot-1",
    "deploymentconfig": "example-spring-boot"
},
...
```

Diese Verknüpfung ist mittels dem `oc describe` Befehl zu sehen:

```
# oc describe service example-spring-boot
```

```
Name:           example-spring-boot
Namespace:      techlab
Labels:         app=example-spring-boot
Selector:       app=example-spring-boot,deploymentconfig=example-spring-boot
Type:           ClusterIP
IP:             172.30.124.20
Port:           8080-tcp    8080/TCP
Endpoints:      10.1.3.20:8080
Session Affinity:   None
No events.
```

Unter Endpoints findet man nun den aktuell laufenden Pod.

ImageStream

ImageStreams werden dafür verwendet, automatische Tasks auszuführen wie bspw. ein Deployment zu aktualisieren, z.B. wenn eine neue Version des Images oder des Basisimages verfügbar ist.

Builds und Deployments können Image Streams beobachten und auf Änderungen entsprechend reagieren. In unserem Beispiel wird der Image Stream dafür verwendet, ein Deployment zu triggern, sobald etwas am Image geändert hat.

Mit dem folgenden Befehl können zusätzliche Informationen über den Image Stream ausgelesen werden:

```
# oc get imagestream example-spring-boot -o json
```

DeploymentConfig

In der DeploymentConfig werden folgende Punkte definiert:

- **Update Strategy**: wie werden Applikationsupdates ausgeführt, wie erfolgt das Austauschen der Container?
- **Triggers**: Welche Triggers führen zu einem Deployment? In unserem Beispiel ImageChange
- **Container**
  - Welches Image soll deployed werden?
  - Environment Configuration für die Pods
  - ImagePullPolicy
- **Replicas**, Anzahl der Pods, die deployt werden sollen

Mit dem folgenden Befehl können zusätzliche Informationen zur DeploymentConfig ausgelesen werden:

```
# oc get deploymentConfig example-spring-boot -o json
```

Im Gegensatz zur DeploymentConfig, mit welcher man OpenShift sagt, wie eine Applikation deployt werden soll, definiert man mit dem ReplicationController, wie die Applikation während der Laufzeit aussehen soll (bspw. dass immer 3 Replicas laufen sollen).

Tipp: für jeden Resource Type gibt es auch eine Kurzform. So könnte man bspw. `oc get deploymentconfig` auch einfach als `oc get dc` schreiben.

Zusatzaufgabe: LAB 3.2

Schaue dir die erstellten Ressourcen mit `oc get [ResourceType] [Name] -o json` und `oc describe [ResourceType] [Name]` aus dem ersten Projekt `[DEINNAME]-example1` an.

Ende von Lab 3! Weiter mit Lab 4

Last update: **2018/07/11 15:39**