Laboratuvar - ConfigMap'leri, DaemonSet'leri, Kubernetes Servislerini, Gizli Anahtarları ve Kalıcı Hacim Taleplerini Anlamak

Gerekli tahmini süre: 1 saat

Bu laboratuvar, Kubernetes'e bir uygulama oluşturup dağıtmanızı, ardından ConfigMap'leri, DaemonSet'leri, Kubernetes Servislerini, Gizli Anahtarları anlamanızı ve oluşturmanızı, ayrıca Hacimler ve Kalıcı Hacim Taleplerini daha derinlemesine keşfetmenizi sağlayacaktır.

Amaçlar

Bu Uygulama Projesi'nde, Docker kullanarak bir JavaScript uygulamasını Kubernetes'e oluşturup dağıtacaksınız.
Aşağıdakileri anlayacak ve oluşturacaksınız:

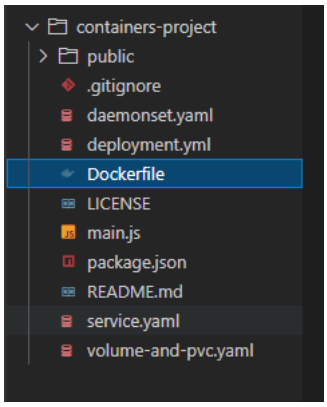
- ConfigMap
- DaemonSet
- Kubernetes Servisi
- Gizli Anahtar
- Hacimler ve Kalıcı Hacim Talepleri

Dockerfile Hakkında

1. Projeye başlamak için başlangıç kodunu içeren depoyu klonlayın.

   ```
   git clone https://github.com/ibm-developer-skills-network/containers-project.git
   ```

2. Ana proje dizininde bulunan `Dockerfile` dosyasını açın.



3. İçeriği aşağıdaki gibi olacaktır:

   ```
   # Use an official Node.js runtime as a parent image
   FROM node:14
   # Set the working directory in the container
   WORKDIR /app
   # Copy the application files to the working directory
   COPY main.js .
   COPY public/index.html public/index.html
   COPY public/style.css public/style.css
   # Make port 3000 available to the world outside this container
   EXPOSE 3000
   # Run the application when the container launches
   CMD ["node", "main.js"]
   ```

İşte kodun açıklaması:

1. `FROM node:14` kullanılacak temel imajı belirtir, bu resmi Node.js çalışma zamanı sürüm 14'tür.
2. `WORKDIR /app` konteyner içindeki çalışma dizinini /app olarak ayarlar.
3. `COPY main.js .` main.js dosyasını ana makineden konteynerdeki mevcut dizine (.) kopyalar.
4. `COPY public/index.html public/index.html` ana makinedeki public dizininden index.html dosyasını konteynerdeki public dizinine kopyalar.
5. `COPY public/style.css public/style.css` ana makinedeki public dizininden style.css dosyasını konteynerdeki public dizinine kopyalar.
6. `EXPOSE 3000` konteynerin 3000 numaralı portunu dışarıdan bağlantılara açar.
7. `CMD ["node", "main.js"]` konteyner başlatıldığında çalıştırılacak komutu belirtir, bu da node main.js'i çalıştırmaktır.

Uygulamayı Kubernetes'e Oluşturma ve Dağıtma

Depo, önceki bölümde gözlemlediğiniz gibi uygulamanın kodunu zaten içeriyor. Sadece docker imajını oluşturup kayıt defterine iteceğiz.

Kubernetes'e dağıtılan uygulamanıza `myapp` adını vereceksiniz.

1. Proje dizinine gidin.

```
cd containers-project/
```

2. Ad alanınızı dışa aktarın.

```
export MY_NAMESPACE=sn-labs-$USERNAME
```

3. Docker imajını oluşturun.

```
docker build . -t us.icr.io/$MY_NAMESPACE/myapp:v1
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ docker build . -t us.icr.io/$MY_NAMESPACE/myapp:v1
[+] Building 78.4s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 464B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:14
 => [auth] library/node:pull token for registry-1.docker.io
 => [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
 => => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
 => => sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3b0fbfc7c05cca5 7.86MB / 7.86MB
 => => sha256:1d12470fa662a2a5cb50378dcdc8ea228c1735747db410bbefb8e2d9144b5452 7.51kB / 7.51kB
 => => sha256:2cafa3fbb0b6529ee4726b4f599ec27ee557ea3dea7019182323b3779959927f 2.21kB / 2.21kB
 => => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca845ebc747976c408 50.45MB / 50.45MB
 => => sha256:3d2201bd995cccf12851a50820de03d34a17011dcbb9ac9fdf3a50c952cbb131 10.00MB / 10.00MB
 => => sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa 776B / 776B
 => => sha256:1de76e268b103d05fa8960e0f77951ff54b912b63429c34f5d6adfd09f5f9ee2 51.88MB / 51.88MB
 => => sha256:d9a8df5894511ce28a05e2925a75e8a4acbd0634c39ad734fdfba8e23d1b1569 191.85MB / 191.85MB
 => => sha256:6f51ee005deac0d99898e41b8ce60ebf250ebe1a31a0b03f613aec6bbc9b83d8 4.19kB / 4.19kB
 => => extracting sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca845ebc747976c408
 => => sha256:5f32ed3c3f278edda4fc571c880b5277355a29ae8f52b52cdf865f058378a590 35.24MB / 35.24MB
 => => sha256:0c8cc2f24a4dcb64e602e086fc9446b0a541e8acd9ad72d2e90df3ba22f158b3 2.29MB / 2.29MB
 => => sha256:0d27a8e861329007574c6766fba946d48e20d2c8e964e873de352603f22c4ceb 450B / 450B
 => => extracting sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3b0fbfc7c05cca5
```

4. Etiketlenmiş görüntüyü IBM Cloud konteyner kayıt defterine gönderin.

```
docker push us.icr.io/$MY_NAMESPACE/myapp:v1
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ docker push us.icr.io/$MY_NAMESPACE/myapp:v1
The push refers to repository [us.icr.io/sn-labs-nikeshkr/myapp]
9d70a531f5e7: Pushed
69bde8258173: Pushed
f64b818c1238: Pushed
a78dd8fb16a7: Pushed
0d5f5a015e5d: Pushed
3c777d951de2: Pushed
f8a91dd5fc84: Pushed
cb81227abde5: Pushed
e01a454893a9: Pushed
c45660adde37: Pushed
fe0fb3ab4a0f: Pushed
f1186e5061f2: Pushed
b2dba7477754: Pushed
v1: digest: sha256:1da35085f4ac8c563114646c6113c2c6f3d1254c0c37c4b05a06ffaa7cba46d7 size: 3042
```

5. Mevcut olan tüm görüntüleri listeleyin. Yeni oluşturulan `myapp` görüntüsünü göreceksiniz.

```
ibmcloud cr images
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ ibmcloud cr images
Listing images...

Repository                                                          Tag
          Digest         Namespace          Created         Size     Security status
us.icr.io/sn-labs-nikeshkr/myapp                                    v1
          1da35085f4ac   sn-labs-nikeshkr   5 minutes ago   350 MB   -
us.icr.io/sn-labsassets/categories-watson-nlp-runtime               latest
          6b01b1e5527b   sn-labsassets      2 years ago     3.1 GB   -
us.icr.io/sn-labsassets/classification-watson-nlp-runtime           latest
          dbd407898549   sn-labsassets      2 years ago     4.0 GB   -
us.icr.io/sn-labsassets/concepts-watson-nlp-runtime                 latest
```

6. Ana proje dizininde bulunan `deployment.yml` dosyasını açın. İçeriği aşağıdaki gibi olacaktır:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: myapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: us.icr.io/<SN-NAMESPACE>/myapp:v1
        imagePullPolicy: Always
        name: myapp
        ports:
        - containerPort: 3000
          name: http
        resources:
          limits:
            cpu: 50m
          requests:
            cpu: 20m
        securityContext:
          allowPrivilegeEscalation: false
          runAsNonRoot: true
          capabilities:
            drop: ["ALL"]
          seccompProfile:
            type: "RuntimeDefault"
          runAsUser: 999
```

İşte içindeki kodun açıklaması.

- `apiVersion`: apps/v1, kullanılan Kubernetes API'sinin sürümünü ve kaynak türünü (Deployment) belirtir.
- `kind`: Deployment, bu YAML'nin bir Deployment nesnesi tanımladığını gösterir.
- `metadata` bölümü, Deployment hakkında ad ve etiketler gibi meta verileri içerir.
- `spec` bölümü, Deployment için istenen durumu tanımlar; bu, kopya sayısını, güncelleme stratejisini ve pod şablonunu içerir.
- `replicas`: 1, uygulamanın çalışması gereken bir kopya olduğunu belirtir.
- `selector`, Deployment'ın hangi Pod'ları yöneteceğini bulmak için etiketleri kullanarak tanımlar.
- `strategy`, Deployment için güncelleme stratejisini belirtir; burada belirli kısıtlamalarla birlikte kademeli güncellemeler kullanılır.
- `template`, yeni Pod'lar oluşturmak için kullanılan Pod şablonunu tanımlar.
- `containers`, Pod içindeki konteynerleri listeler.
- `image`, konteyner için kullanılacak Docker imajını belirtir.
- `imagePullPolicy`: Always, en son imajın her zaman kayıt defterinden çekilmesini sağlar.
- `name`, konteyner için bir ad atar.
- `ports` bölümü, konteyner tarafından hangi portların açılması gerektiğini belirtir.
- `resources`, konteyner için CPU gibi kaynak taleplerini ve sınırlamalarını tanımlar.
- `securityContext`, konteynerin daha iyi güvenlik için kısıtlı, kök olmayan izinlerle çalışmasını sağlar.

7. `<your SN labs namespace>` kısmını gerçek SN lablar ad alanınızla değiştirin.

▼ Ad alanınızı almak için buraya tıklayın

1. `oc project` komutunu çalıştırın ve proje adınıza bağlı olan ad alanını kullanın.

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ oc project
Using project "sn-labs-nikeshkr" from context named "nikeshkr-context" on server "https://c1.us-east.containers.cloud.ibm.
```

2. `ibmcloud cr namespaces` komutunu çalıştırın ve sn-labs-kullanıcı adınızı gösteren olanı kullanın.

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ ibmcloud cr namespaces
Listing namespaces for account 'QuickLabs - IBM Skills Network' in registry 'us.icr.io'...

Namespace
sn-labs-nikeshkr
sn-labsassets

OK
theia@theiadocker-nikeshkr:/home/project/containers-project$
```

8. Dağıtımı uygulayın.

```
kubectl apply -f deployment.yml
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl apply -f deployment.yml
deployment.apps/myapp created
```

9. Uygulama pod'larının çalıştığını ve erişilebilir olduğunu doğrulayın.

```
kubectl get pods
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
myapp-6c9c7b7cf9-kg4px   1/1     Running   0          15s
```

10. Uygulamayı port yönlendirmesi ile başlatın:

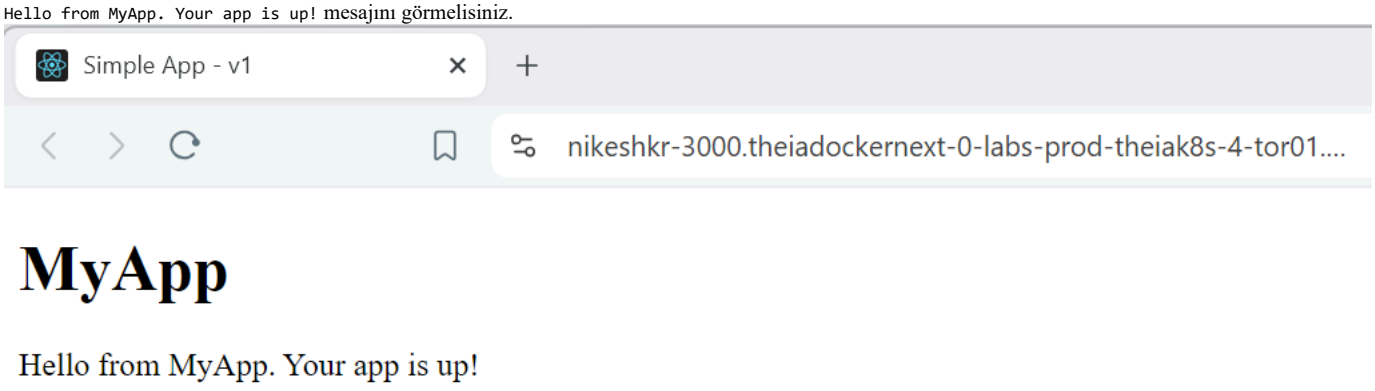
```
kubectl port-forward deployment.apps/myapp 3000:3000
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl port-forward deployment.apps/myapp 3000:3000

Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
```

11. Uygulamayı `3000` numaralı portta başlatın ve uygulama çıktısını görüntüleyin.

12. `Hello from MyApp. Your app is up!` mesajını görmelisiniz.



13. İlerlemeye devam etmeden önce sunucuyu durdurmak için `CTRL + C` tuşlarına basın.

Alıştırma 1: ConfigMap

Bu alıştırmada, myapp uygulaması için yapılandırma verilerini yönetmek üzere bir ConfigMap nasıl kurulacağını öğreneceksiniz.

1. Aşağıdaki komut, `myapp-config` adında bir ConfigMap oluşturmak için kullanılan sözdizimidir (yer tutucu değerlerle).

```
kubectl create configmap myapp-config --from-literal=env-var1=value1 --from-literal=env-var2=value2
```

- Bu komut, myapp uygulaması için ortamla ilgili yapılandırma verilerini saklayan `myapp-config` adlı bir ConfigMap oluşturur.
- `--from-literal` bayrağı, ConfigMap için verilerin doğrudan komut satırında sağlanacağını belirtir.
- Anahtar-değer çiftlerini tanımlar; bu çiftler, ConfigMap içindeki ortam değişkenlerini ve bunların karşılık gelen değerlerini belirler.

2. Örnek olarak, aşağıdaki anahtar-değer çiftlerini kullanabilirsiniz:
env-var1: `server-url` ; value1: `http://example.com`
env-var2: `timeout` ; value2: `5000`

3. Bunlara göre, bir configmap oluşturmak için aşağıdaki komutu çalıştırın:

```
kubectl create configmap myapp-config --from-literal=server-url=http://example.com --from-literal=timeout=5000
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl create configmap myapp-config --from-literal
=server-url=http://example.com --from-literal=timeout=5000
configmap/myapp-config created
```

- Burada, **server-url** `http://example.com` olarak ayarlanmış ve **timeout** `5000` olarak ayarlanmıştır.
- **server-url**, uygulamanın iletişim kurması gereken bir dış sunucunun URL'sini saklayabilir ve **timeout**, uygulamanın zaman aşımına uğramadan önce bir yanıt beklediği maksimum süreyi temsil edebilir.

4. Aşağıdaki komutu çalıştırarak ConfigMap'in başarılı bir şekilde oluşturulduğunu doğrulayın:

```
kubectl get configmap myapp-config
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl get configmap myapp-config
NAME           DATA   AGE
myapp-config   2      21s
```

- Buradaki çıktı, `myapp-config` adında bir ConfigMap'in varlığını gösteriyor. Bu ConfigMap, iki anahtar-değer çifti içeriyor ve `26 saniye` önce oluşturulmuş.

Egzersiz 2: DaemonSet'ler

Bu egzersizde, 'myapp' pod'larının dağıtıldığı düğümler de dahil olmak üzere, kümedeki her düğümde bir pod'un çalışmasını sağlamak için bir DaemonSet oluşturacaksınız. Bir DaemonSet oluşturmak, uygulamanızın (`myapp`) her düğümde çalışmasını sağlayarak kullanılabilirliğini ve hata toleransını artırır, bu da yedeklilik ve yük dağılımı sağlar.

1. Ana proje dizininde bulunan `daemonset.yaml` dosyasını açın. İçeriği aşağıdaki gibi olacaktır:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: myapp-daemonset
  labels:
    app: myapp
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: us.icr.io/<SN-NAMESPACE>/myapp:v1
        ports:
        - containerPort: 3000
          name: http
        securityContext:
          allowPrivilegeEscalation: false
          runAsNonRoot: true
          capabilities:
            drop: ["ALL"]
          seccompProfile:
            type: "RuntimeDefault"
          runAsUser: 999
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

Aşağıda kodun açıklaması verilmiştir:

- `apiVersion: apps/v1`: Bu satır, bu YAML dosyasının uyduğu Kubernetes API sürümünü belirtir. Bu durumda, Kubernetes uygulamaları için API sürümü olan `apps/v1` kullanılır.

- `kind: DaemonSet`: Bu satır, bu YAML dosyasında tanımlanan Kubernetes kaynak türünü belirtir. Bu durumda, bir DaemonSet'tir. DaemonSet, tüm (veya bazı) düğümlerin belirli bir pod'un bir kopyasını çalıştırmasını sağlar.

```
metadata:
  name: myapp-daemonset
  labels:
    app: myapp
```

- Yukarıdaki satırlar DaemonSet hakkında meta verileri belirtir. DaemonSet'e bir ad (`myapp-daemonset`) verir ve ona etiketler ekler. Etiketler, nesnelerin alt kümlerini düzenlemek ve seçmek için kullanılan anahtar-değer çiftleridir. Bu durumda, `app: myapp` etiketi bu DaemonSet'e eklenmiştir.

```
spec:
  selector:
    matchLabels:
      app: myapp
```

- Yukarıdaki satırlar, DaemonSet için seçiciyi tanımlar. DaemonSet'in hangi pod'ları yönetmesi gerektiğini nasıl belirlediğini belirtir. Burada, `app: myapp` etiketine sahip pod'ları seçer.

```
    spec:
      containers:
      - name: myapp-container
        image: us.icr.io/<your SN labs namespace>/myapp:v1
        ports:
        - containerPort: 3000
          name: http
```

- Yukarıdaki satırlar, podlar içindeki konteynerler için spesifikasyonu tanımlar. Konteynerin adını (`myapp-container`), kullanılacak Docker imajını ve açığa çıkarılacak portları belirtir. Bu durumda, `http` adıyla `3000` portunu açığa çıkarır.

```
        securityContext:
          allowPrivilegeEscalation: false
          runAsNonRoot: true
          capabilities:
            drop: ["ALL"]
          seccompProfile:
            type: "RuntimeDefault"
          runAsUser: 999
```

- Yukarıdaki satırlar, konteyner için güvenlik bağlamını tanımlar. Pod'un, sınırlı ayrıcalıklarla kök olmayan bir kullanıcı olarak çalışmasını sağlar, tüm yetenekleri kaldırır, ayrıcalık yükseltmesini engeller ve daha güvenli konteyner çalıştırma için varsayılan seccomp profilini uygular.

```
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

- Yukarıdaki satırlar, DaemonSet pod'ları için toleransları tanımlar. Toleranslar, pod'ların eşleşen taint'lere sahip düğümlere planlanmasına izin verir. Burada, `node-role.kubernetes.io/master` anahtarına sahip taint'i ve `NoSchedule` etkisini tolere eder, bu da onun master rolüne sahip düğümlere planlanabileceği anlamına gelir.

2. Aşağıdaki komutu kullanarak `daemonset.yaml` dosyasını Kubernetes kümenize uygulayın:

```
kubectl apply -f daemonset.yaml
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl apply -f daemonset.yaml
daemonset.apps/myapp-daemonset created
```

3. DaemonSet'i uyguladıktan sonra, başarılı bir şekilde dağıtıldığını ve beklendiği gibi çalıştığını doğrulamak için durumunu kontrol edin. DaemonSet'lerin durumunu kontrol etmek için aşağıdaki komutu kullanın:

   ```
   kubectl get daemonsets
   ```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl get daemonsets
NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
myapp-daemonset   6         6         6       6            6           <none>          23s
```

Bu komut, Kubernetes kümesindeki DaemonSet'ler hakkında bilgi gösterir.

Örneğin, sağlanan çıktıda:

`DESIRED : 6` DaemonSet pod'larının istenen sayısını belirtir.
`CURRENT : 6` mevcut DaemonSet pod'larının sayısını ifade eder.
`READY : 6` hazır olan DaemonSet pod'larının sayısını temsil eder.
`UP-TO-DATE : 6` tüm DaemonSet pod'larının güncel olduğunu ima eder.
`AVAILABLE : 6` tüm DaemonSet pod'larının hizmet için mevcut olduğunu gösterir.
`NODE SELECTOR : <none>` hiçbir düğüm seçicisinin belirtilmediği anlamına gelir.
`AGE : 23s` DaemonSet'in oluşturulmasından bu yana geçen süreyi, yani **23 saniye**yi belirtir.

Alıştırma 3: Kubernetes hizmetleri

Bu alıştırmada, uygulamanızı küme içinde açığa çıkarmak için bir Kubernetes Hizmeti oluşturacaksınız.

1. Ana proje dizininde bulunan `service.yaml` dosyasını açın. İçeriği aşağıdaki gibi olacaktır:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: myapp-service
   spec:
     selector:
       app: myapp
     ports:
       - protocol: TCP
         port: 80
         targetPort: 3000
     type: NodePort
   ```

Aşağıda kodun açıklaması verilmiştir:

`apiVersion: v1`: Bu satır, bu YAML dosyasının uyduğu Kubernetes API sürümünü belirtir. Bu durumda, en temel Kubernetes API sürümü olan **core/v1** API sürümü kullanılmaktadır.

`kind: Service`: Bu satır, bu YAML dosyasında tanımlanan Kubernetes kaynak türünü belirtir. Bu durumda, bir Service'dir. Kubernetes'te bir Service, mantıksal bir pod kümesini tanımlayan ve bunlara erişim için bir politika belirleyen bir soyutlamadır.

```
metadata:
  name: myapp-service
```

- Yukarıdaki satırlar, Servis hakkında bir ad (`myapp-service`) vererek meta verileri belirtir.

```
spec:
  selector:
    app: myapp
```

- Bu satırlar, Servis için seçiciyi tanımlar ve Servisin hangi pod'ları etiketlerine göre hedeflemesi gerektiğini belirtir. Bu durumda, `app: myapp` etiketine sahip pod'ları seçer.

```
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
```

- Bu satırlar, Servis için port yapılandırmasını tanımlar. Servis üzerinde açılacak portları ve trafiğin nereye yönlendirileceğini belirtir. Burada, Servis üzerinde 80 numaralı portu açar ve trafiği pod'lara 3000 numaralı porta yönlendirir.

```
  type: NodePort
```

- Bu satır, Servis türünü `NodePort` olarak belirtir. Bu, Servisi, kümedeki her bir düğümün belirli bir portunda istemcilerin erişebilmesi için açan bir Kubernetes Servisidir.

2. Bu yapılandırmayı Kubernetes kümenize uygulayın.

```
kubectl apply -f service.yaml
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl apply -f service.yaml
service/myapp-service created
```

3. Mevcut tüm hizmetleri alın.

```
kubectl get services
```

```
theia@theiadocker-nikeshkr:/home/project/containers-project$ kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
myapp-service   NodePort    172.21.26.244   <none>        80:30016/TCP   25s
```

- Bu komut, hem lab ortamında zaten mevcut olan `openshift-web-console` hizmetini hem de yeni oluşturduğunuz myapp-service'i görüntüleyecektir.

Örneğin, `myapp-service` için sağlanan çıktıda:

`TYPE: NodePort`: Bu hizmetin NodePort türünde olduğunu belirtir; yani hizmet, kümedeki tüm düğümlerde bir port üzerinden erişilebilir.

`CLUSTER-IP: 172.21.26.244`: Bu, hizmete atanan iç Cluster IP adresidir. Kubernetes kümesi içinde iletişim için kullanılır.

`EXTERNAL-IP: <none>`: Bu, bu hizmete atanmış bir dış IP adresi olmadığını gösterir. Dış istemciler, bu hizmete Kubernetes kümesinin dışından doğrudan erişemez.

`PORT(S): 80:30016/TCP`: Bu, **80** numaralı portun hizmet tarafından dahili olarak açıldığını ve dışarıda **30016** numaralı port üzerinden **TCP protokolü** ile erişilebilir olduğunu belirtir. Format `externalPort/protocol` şeklindedir.

`AGE: 25s`: Bu, hizmetin oluşturulmasından itibaren 25 saniye boyunca çalıştığını gösterir.

Not: Cluster-IP adreslerinin veya portların mevcut olmasına bağlı olarak bu değer lab ortamınızda farklılık gösterebilir.

Egzersiz 4: Gizli Bilgiler

Bu egzersizde, şifreler, tokenlar ve SSH anahtarları gibi hassas bilgileri güvenli bir şekilde depolamak için Kubernetes'te gizli bilgiler oluşturmayı ve yönetmeyi öğreneceksiniz.

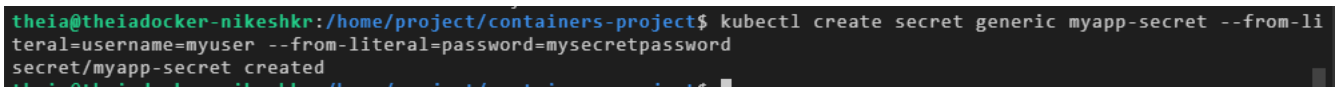
1. Aşağıdaki komut, hassas veriler için anahtar-değer çiftleri sağlayarak `myapp-secret` adında bir gizli bilgi oluşturmanıza yardımcı olur:

   ```
   kubectl create secret generic <secret-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2> ...
   ```

2. Örneğin, aşağıdaki anahtar-değer çiftlerini kullanabilirsiniz:
   anahtar1: `username` ; değer1: `myuser`
   anahtar2: `password` ; değer2: `mysecretpassword`

3. Bunlara göre, bir gizli anahtar oluşturmak için aşağıdaki komutu çalıştırın:

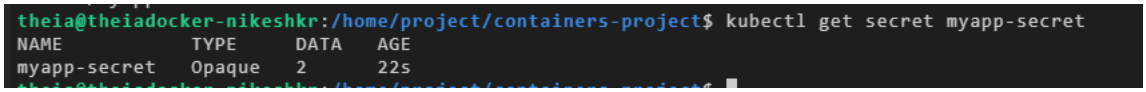
   ```
   kubectl create secret generic myapp-secret --from-literal=username=myuser --from-literal=password=mysecretpassword
   ```



- Bu komut, `myapp-secret` adında bir Secret oluşturur ve bunu `username` değerini `myuser` ve `password` değerini `mysecretpassword` olarak ayarlanan anahtar-değer çiftleri ile doldurur.

4. Secret'ın başarılı bir şekilde oluşturulduğunu doğrulamak için aşağıdaki komutu çalıştırın:

   ```
   kubectl get secret myapp-secret
   ```



Çıktı, `myapp-secret` Secret hakkında aşağıdaki bilgileri gösterir:

`NAME`: Bu sütun, sırasıyla myapp-secret olan sırrın adını gösterir.

`TYPE`: Bu sütun, sırrın türünü belirtir. Burada, **Opaque** olarak gösterir, bu da onun genel bir sır olduğunu ve belirli bir türle ilişkilendirilmediğini belirtir.

`DATA`: Bu sütun, sır içinde saklanan veri girişlerinin sayısını gösterir. Bu durumda, **2** gösterir, bu da sır içinde iki veri parçasının saklandığını belirtir.

`AGE`: Bu sütun, sırın ne zamandan beri oluşturulduğunu veya en son ne zaman güncellendiğini gösterir. Bu durumda, *22 saniye*dir.

Egzersiz 5: Hacimler ve kalıcı hacim talepleri

Bu egzersizde, uygulamanız için depolama sağlamak amacıyla Kubernetes'te hacimleri ve kalıcı hacim taleplerini (PVC) nasıl tanımlayacağınızı keşfedeceksiniz.

Bir `PersistentVolume (PV)`, küme içinde bir yönetici tarafından sağlanan, herhangi bir Pod'dan bağımsız olarak var olan bir depolama kaynağıdır. Dinamik olarak sağlanabilen veya statik olarak tanımlanabilen, küme genelinde bir kaynaktır ve yaşam döngüsü küme yöneticisi tarafından yönetilmektedir.

Bir `PersistentVolumeClaim (PVC)`, kalıcı hacim(ler)i tüketen bir kullanıcı veya Pod tarafından depolama talebidir. PVC'ler, kullanıcıların ihtiyaç duyduğu depolama kaynaklarını talep etmeleri için bir yol sağlar. Namespace'e özgüdürler ve yalnızca kendi namespace'leri içinde depolama talep edebilirler; yaşam döngüleri, onları oluşturan kullanıcı veya geliştirici tarafından yönetilmektedir.

1. **Hem bir PV hem de bir PVC** tanımlayan `volume-and-pvc.yaml` adlı dosyayı keşfedin.

2. Aşağıdaki kod parçası bir PV oluşturur:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myapp-volume
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data
```

İşte kodun bir açıklaması:

- `apiVersion: v1`: Kullanılan Kubernetes API sürümünü belirtir.
- `kind: PersistentVolume`: Bu YAML'nın bir PersistentVolume kaynağını tanımladığını gösterir.
- `metadata`: PersistentVolume hakkında meta verileri içerir.
- `name: myapp-volume`: PersistentVolume'un adını belirtir, bu durumda 'myapp-volume' olarak tanımlanmıştır.
- `spec`: Bu, PersistentVolume'un spesifikasyonunu tanımlar.
- `capacity`: PersistentVolume'un kapasitesini belirtir.
- `storage: 1Gi`: Bu, PersistentVolume'un 1 gigabayt depolama kapasitesine sahip olduğunu gösterir.
- `accessModes`: PersistentVolume için erişim modlarını tanımlar.
    - `ReadWriteOnce`: Hacmin tek bir düğüm tarafından okuma-yazma olarak bağlanabileceğini belirtir.
    - `hostPath`: Bir ana bilgisayar yolu hacim kaynağını belirtir.
    - `path: /data`: PersistentVolume'un ana bilgisayardaki bir dizinle desteklendiğini gösterir (bu durumda /data).

3. Üç tire (yani, `---`), PV tanımını PVC tanımından ayırır.

4. Aşağıdaki kod parçası bir PVC oluşturur:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

İşte kodun açıklaması:

- `apiVersion: v1`: PersistentVolumeClaim için kullanılan Kubernetes API sürümünü belirtir.
- `kind: PersistentVolumeClaim`: Bu YAML'nin bir PersistentVolumeClaim kaynağını tanımladığını gösterir.
- `metadata`: PersistentVolumeClaim hakkında meta verileri içerir.
- `name: myapp-pvc`: PersistentVolumeClaim'in adını belirtir, bu durumda `myapp-pvc`'dir.
- `spec`: PersistentVolumeClaim'in spesifikasyonunu tanımlar.
- `accessModes`: PersistentVolumeClaim için erişim modlarını tanımlar.
    - `ReadWriteOnce`: Hacmin bir düğüm tarafından okuma-yazma olarak bağlanabileceğini belirtir.
    - `resources`: PersistentVolumeClaim için kaynak gereksinimlerini belirtir.
    - `requests`: İstenen kaynakları gösterir.
    - `storage: 1Gi`: PersistentVolumeClaim'in **1 gigabayt** depolama kapasitesi talep ettiğini belirtir.

Sonuç

Bu Pratik Projede, bir Javascript uygulaması oluşturarak ve Docker kullanarak Kubernetes'e dağıtarak başladınız. Uygulamanın yapılandırma verilerini yönetmek için bir ConfigMap oluşturduğunuz ve anladığınız, kümedeki her düğümde bir pod'un çalışmasını sağlamak için bir DaemonSet, uygulamanızı küme içinde dışa açmak için bir Kubernetes Servisi ve hassas bilgileri güvenli bir şekilde depolamak için bir Secret geliştirdiniz. Ayrıca, uygulamanız için depolama sağlamak üzere hacimler ve kalıcı hacim talepleri tanımlamayı keşfettiniz.

Yazar(lar)

K Sundararajan

© IBM Corporation. Tüm hakları saklıdır.