YAML file examples

MN Owned by Mandra NXT •••
Last updated: less than a minute ago

With Multiple Replicas

The following YAML configuration creates a Deployment object that runs 5 replicas of an NGINX container.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 5
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      —name: nginx
        image: nginx
        ports:
        —containerPort: 80
```

With Resource Limits

The following YAML configuration creates a Deployment object similar to the above, but with resource limits.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 5
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      —name: nginx
        image: nginx
```

```
          resources:
            limits:
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          ports:
           —containerPort: 80
```

With Health Checks

The following YAML configuration creates a Deployment object that performs a health check on containers by checking for an HTTP response on the root directory.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 5
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
       —name: nginx
          image: nginx
          ports:
           —containerPort: 80
          livenessProbe:
            httpGet:
              path: /
              port: 80
            initialDelaySeconds: 5
            periodSeconds: 5
```

With Persistent Volumes

The following YAML configuration creates a Deployment object that creates containers that request a PersistentVolume (PV) using a PersistentVolumeClaim (PVC), and mount it on a path within the container.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
```

```yaml
    matchLabels:
      app: web
  replicas: 5
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      volumes:
       —name: my-pv-storage
          persistentVolumeClaim:
            claimName: my-pv-claim
      containers:
       —name: nginx
          image: nginx
          ports:
           —containerPort: 80
          volumeMounts:
           —mountPath: "/usr/share/nginx/html"
              name: my-pv-storage
```

With Affinity Settings

The following YAML configuration creates a Deployment object with affinity criteria that can encourage a pod to schedule on certain types of nodes.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: web
spec:
  selector:
    matchLabels:
      app: web
  replicas: 5
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: web
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
           —matchExpressions:
             —key: disktype
                operator: In
                values:
               —ssd
      containers:
       —name: nginx
          image: nginx
          ports:
           —containerPort: 80
```

With NodePort, exposing to internet

- This type of Service allows external accessibility to a Pod on a node.
- It also exposes an application externally with the help of a NodeIP on which the Pod is running.
- The NodePort range is 30000 – 32767; declaration outside this range is impossible.

Traffic Flow:

Internet > Node External IP: 80 or 443 > Node Internal IP: 30000-32767 > Pod IP: 80 or 443

Step-1 Create a deployment YAML file called `deployment-corpwebsite.yaml`

```
sudo nano deployment-corpwebsite.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-corpwebsite
spec:
  selector:
    matchLabels:
      app: pod-corpwebsite
  replicas: 2
  template:
    metadata:
      labels:
        app: pod-corpwebsite
        env: dev
    spec:
      containers:
      - name: container-corpwebsite
        image: tanvisinghny/ssl-website
        ports:
        - containerPort: 80
        - containerPort: 443
```

Step-2 Apply this file to create deployment

```
kubectl apply -f deployment-corpwebsite.yaml
```

Step-3 Verify the Pods creation

```
linuxadmin@master:~$ kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
deployment-corpwebsite-7675c48cc6-6zn5d   1/1     Running   0          81m
deployment-corpwebsite-7675c48cc6-rl4sh   1/1     Running   0          81m
```

Step-4 Verify Deployment Object creation

```
linuxadmin@master:~$ kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment-corpwebsite   2/2     2            2           24h
```

Step-5 Verify that a Replica Set is created

```
linuxadmin@master:~$ kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
deployment-corpwebsite-7675c48cc6   2         2         2       82m
deployment-corpwebsite-7f948548fd   0         0         0       24h
```

Filter Pods by Label env=dev

```
linuxadmin@master:~$ kubectl get pods -l env=dev
NAME                                      READY   STATUS    RESTARTS   AGE
deployment-corpwebsite-7675c48cc6-6zn5d   1/1     Running   0          83m
deployment-corpwebsite-7675c48cc6-rl4sh   1/1     Running   0          83m
```

Step-6: Create a service YAML file:

```
sudo nano service-corpwebsite.yaml
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-corpwebsite
spec:
  type: NodePort
  selector:
    app: pod-corpwebsite
  ports:
    - nodePort: 30163
      port: 443
      targetPort: 443
  externalIPs:
    - 10.138.0.21
    - 10.138.0.22

```

External IPs are Worker Node IPs:

- Node1 Internal IP = `10.138.0.21`
- Node2 Internal IP = `10.138.0.22`

Apply the Service:

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

Find if the Service is created:

```
linuxadmin@master:~$ kubectl get svc
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP                   PORT(S)         AGE
kubernetes            ClusterIP   10.96.0.1       <none>                        443/TCP         36h
service-corpwebsite   NodePort    10.109.37.230   10.138.0.21,10.138.0.22   443:30163/TCP   22h
```