

DEPLOYMENT OF APP IN IBM CLOUD

Project Date	September 2022
Team ID	PNT2022TMID30533
Project Name	Containment Zone Alerting Application

Deploy in Kubernetes Cluster:

What is Kubernetes Deployment YAML?

YAML (which stands for Y Ain't Markup Language) is a language used to provide configuration for software, and is the main type of input for Kubernetes configurations. It is human-readable and can be authored in any text editor.

A Kubernetes user or administrator specifies data in a YAML file, typically to define a Kubernetes object. The YAML configuration is called a “manifest”, and when it is “applied” to a Kubernetes cluster, Kubernetes creates an object based on the configuration.

A Kubernetes Deployment YAML specifies the configuration for a Deployment object—this is a Kubernetes object that can create and update a set of identical pods. Each pod runs specific containers, which are defined in the spec.template field of the YAML configuration.

The Deployment object not only creates the pods but also ensures the correct number of pods is always running in the cluster, handles scalability, and takes care of updates to the pods on an ongoing basis. All these activities can be configured through fields in the Deployment YAML.

Below we'll show several examples that will walk you through the most common options in a Kubernetes Deployment YAML manifest.

Kubernetes Deployment YAML Examples

With Multiple Replicas

The following YAML configuration creates a Deployment object that runs 5 replicas

of an NGINX container.

```
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
```

Important points in this configuration:

- spec.replicas

—specifies how many pods to run

- strategy.type

—specifies which deployment strategy should be used. In this case and in the following examples we select RollingUpdate, which means new versions are rolled out gradually to pods to avoid downtime.

- spec.template.spec.containers

—specifies which container image to run in each of the pods and ports to expose.

With Resource Limits

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

```
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: 8

```

The `spec.containers.resources` field specifies:

- limits

—each container should not be allowed to consume more than 200Mi of memory.

- requests

—each container requires 100m of CPU resources and 200Mi of memory on the node

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

```

The `template.spec.containers.livenessProbe` field defines what the kubelet should check to ensure

that the pod is alive:

- `httpGet`

specifies that the kubelet should try a HTTP request on the root of the web server on

port 80.

- `periodSeconds`

specifies how often the kubelet should perform a liveness probe.

- `initialDelaySeconds`

specifies how long the kubelet should wait after the pod starts, before performing the first probe.

You can also define readiness probes and startup probes—learn more in

the [Kubernetes documentation](#).

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: 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
```

Important points in this configuration:

- `template.spec.volumes`

—defines a name for the volume, which is referenced below in

`containers.volumeMounts`

- `template.spec.volumes.persistentVolumeClaim`

—references a PVC. For this to work, you must have some PVs in your cluster and

create a PVC object that matches those PVs. You can then reference the existing PVC

object here and the pod will attempt to bind to a matching PV.

Learn more about PVs and PVCs in the [documentation](#).

With Affinity Settings

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

object with affinity
criteria

```
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
```

The

spec.affinity

field defines criteria that can affect whether the pod schedules on a certain node or not:

- spec.affinity.nodeAffinity

—specifies desired criteria of a node which will cause the pod to be scheduled on it

- `spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution`

—specifies that affinity is relevant when scheduling a new pod, but is ignored when the pod is already running.

- `nodeSelectorTerms`

—specifies, in this case, that the node needs to have a disk of type SSD for the pod to be scheduled.

There are many other options, including preferred node affinity, and pod affinity, which means the pod is scheduled based on the criteria of other pods running on the same node. Learn more in the [documentation](#).