**What is a Kubernetes Deployment YAML file?**

A **Kubernetes Deployment YAML** file is a configuration file written in [YAML (YAML Ain’t Markup Language)](https://spacelift.io/blog/yaml) that defines the desired state of a Kubernetes Deployment. This YAML file is used to create, update, or delete Deployments in a Kubernetes cluster. It contains a set of key-value pairs that specify various attributes and settings for the Deployment, such as the number of replicas, pod template specifications, labels, and more.

**How do I create a YAML file for Kubernetes deployment?**

Create a new file and give it the extension .yaml (e.g. deployment.yaml).

You can also create or apply a YAML file using the kubectl apply -f <filename.yaml> command, which will create the Deployment in your Kubernetes cluster according to the specifications defined in the YAML file. Similarly, you can use YAML files to update or delete Deployments, depending on the changes you want to make to the cluster’s desired state.

A typical Kubernetes Deployment YAML file includes the following key components:

* **apiVersion**: Specifies the Kubernetes API version, such as “apps/v1” for Deployments.
* **kind**: Specifies the type of Kubernetes resource, in this case, “Deployment.”
* **metadata**: Provides metadata for the Deployment, including the name, labels, and annotations.
* **spec**: Defines the desired state of the Deployment, including the number of replicas, the pod template, and any other related specifications. It includes:
* **replicas**: Specifies the desired number of identical pod replicas to run.
* **selector**: Specifies the labels that the Replica Set uses to select the pods it should manage.
* **template**: Contains the pod template used for creating new pods, including container specifications, image names, and container ports.

**Example of a basic YAML file for Kubernetes deployment**

The example below will create a Kubernetes Deployment named “example-deployment” with three replicas, each running a pod based on the specified container image and port configuration. The pods will be labeled with “app: example,” and they will be managed by the Deployment.

apiVersion: apps/v1

kind: Deployment

metadata:

name: example-deployment

spec:

replicas: 3

selector:

matchLabels:

app: example

template:

metadata:

labels:

app: example

spec:

containers:

- name: example-container

image: example-image

ports:

- containerPort: 8080

Let’s break each line down…

* apiVersion: Specifies the Kubernetes API version. In this case, it’s using the “apps/v1” API version, which is appropriate for Deployments.
* kind: Specifies the type of Kubernetes resource. Here, it’s “Deployment,” indicating that this configuration file is defining a Deployment.
* spec: This section defines the desired state of the Deployment.
* replicas: 3: Specifies that you want to run three replicas of your application.
* selector: Describes the selector to match pods managed by this Deployment.
* matchLabels: Specifies the labels that the Replica Set created by the Deployment should use to select the pods it manages. In this case, pods with the label app: example are selected.
* template: Defines the pod template used for creating new pods.
* metadata: Contains the labels to apply to the pods created from this template. In this case, the pods will have the label app: example.
* spec: Describes the specification of the pods.
* containers: This section specifies the containers to run in the pod.
* name: example-container: Assigns a name to the container.
* image: example-image: Specifies the Docker image to use for this container.
* ports: Defines the ports to open in the container.
* containerPort: 8080: Indicates that the container will listen on port 80.