📘 Understanding Pods & Exploring Deployment Features: Auto-Scaling, Rolling Updates, and Self-Healing with Manifest YAML Files and kubectl Commands 🚀🔧

🌐 Overview of Kubernetes Deployment

Kubernetes Deployments provide advanced features like **auto-healing**, **rolling updates**, and **auto-scaling**, which are not available when deploying Pods directly. This guide demonstrates how to create and manage Kubernetes Pods and Deployments with YAML manifests and relevant kubectl commands to create, update, and validate resources.

Before jumping into Deployments, let's understand what a Pod is:

What is a Pod?

A Pod defines how containers should be set up and run within a cluster. It is the basic unit in Kubernetes that groups one or more containers, sharing the same network and storage. This setup allows containers within the same Pod to communicate easily and work together.

Why Use Deployments Instead of Pods?

Although a **Pod** is the basic unit in Kubernetes, managing Pods directly comes with limitations. Pods do not have built-in mechanisms for recovery if they fail, nor do they support advanced features like automatic scaling, rolling updates, or easy rollback capabilities.

Deployments offer significant advantages over standalone Pods:

- **Automatic Scaling**: With Deployments, you can automatically scale the number of Pods based on resource utilization or demand.
- **Self-Healing**: Deployments will replace failed Pods, ensuring that the desired number of Pods is always maintained without manual intervention.
- **Rolling Updates & Rollbacks**: Deployments allow for seamless rolling updates, ensuring zero downtime during version updates. Additionally, you can easily roll back to previous versions in case of failures.
- **Replica Management**: Deployments manage multiple Pod replicas to provide high availability and load balancing across your application.

In short, Deployments are essential for ensuring your applications are resilient, scalable, and easier to manage, especially in production environments.

Let’s understand how to create a Pod and a Deployment, and explore Deployment features such as auto-scaling, rolling updates, and self-healing using manifest YAML files and kubectl commands in the following steps.

🛠️ Step 1: Create a Pod

This manifest defines a simple Pod running an NGINX container.

Pod Manifest (nginx-pod.yaml)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.19
      ports:
        - containerPort: 80
```

kubectl Commands:

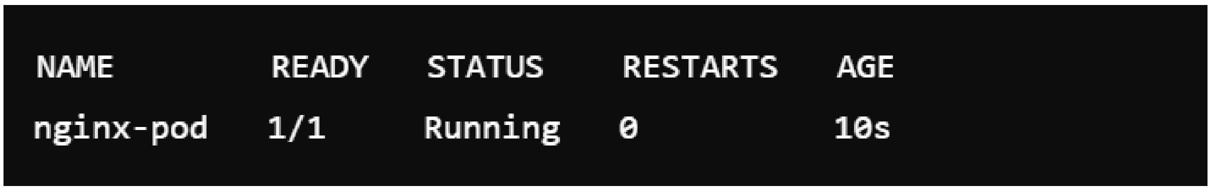
1. **Create the Pod**:

```bash
kubectl apply -f nginx-pod.yaml
```

2. **Validate Pod Creation:**

```bash
kubectl get pods
```

You should see the **nginx-pod** in the list of running Pods:



```
NAME         READY    STATUS     RESTARTS    AGE
nginx-pod    1/1      Running    0           10s
```

3. **Check Pod Details:**

```bash
kubectl describe pod nginx-pod
```

🛠 Step 2: Create a Deployment

Deployment Manifest (nginx-deployment.yaml)

The following manifest defines a Deployment that manages 3 replicas of NGINX Pods.

nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3  # Number of pod replicas
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.19
          ports:
            - containerPort: 80
```

kubectl Commands:

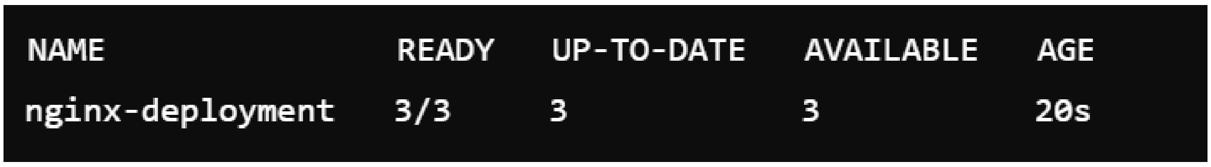
1. **Create the Deployment**:

```bash
kubectl apply -f nginx-deployment.yaml
```

2. **Validate Deployment Creation**:

```bash
kubectl get deployments
```

You should see the **nginx-deployment** listed:

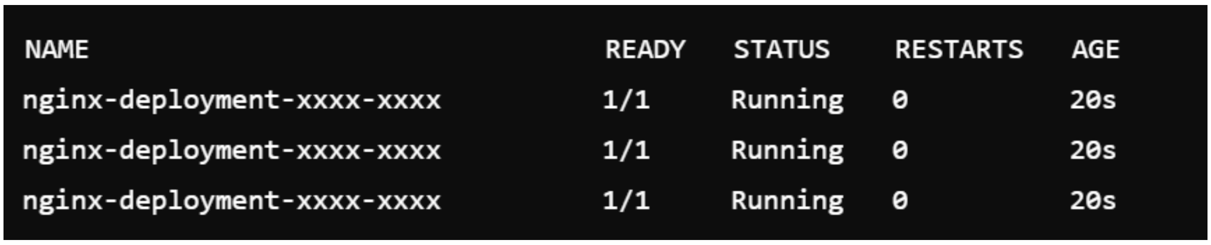


```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           20s
```

3. **Check Pod Replicas**:

```bash
kubectl get pods
```

This will show 3 Pods created by the Deployment:



```
NAME                            READY   STATUS    RESTARTS   AGE
nginx-deployment-xxxx-xxxx      1/1     Running   0          20s
nginx-deployment-xxxx-xxxx      1/1     Running   0          20s
nginx-deployment-xxxx-xxxx      1/1     Running   0          20s
```

4. **Describe Deployment for Details**:

```bash
kubectl describe deployment nginx-deployment
```

🛠️ Step 3: Rolling Update for the Deployment

To update the NGINX image version from 1.19 to 1.20, modify the **nginx-deployment.yaml** file:

Updated Deployment Manifest (Rolling Update)

nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

```
    app: nginx
  spec:
   containers:
    - name: nginx
      image: nginx:1.20  # Updated image version
      ports:
       - containerPort: 80
```

kubectl Commands:

1. **Apply the Updated Deployment:**

```bash
kubectl apply -f nginx-deployment.yaml
```

2. **Monitor the Rolling Update:**

```bash
kubectl rollout status deployment/nginx-deployment
```

This command will show the status of the rollout. Once complete, the new Pods will run with the updated NGINX version.

3. **Validate the Update:**

```bash
kubectl get pods
```

You will see new Pods with the updated version running.

4. **Check the Deployment Revision History:**

```bash
kubectl rollout history deployment/nginx-deployment
```

🛠️ Step 4: Auto-Scaling the Deployment

To enable auto-scaling, we'll create a Horizontal Pod Autoscaler (HPA) that scales the number of Pods based on CPU usage. Before setting up HPA, ensure that the Metrics Server is installed and running in your cluster, as it provides the necessary metrics for HPA to function.

Prerequisite: Install Metrics Server

1. **Install Metrics Server**: Metrics Server collects resource metrics from Kubelets and exposes them via the Kubernetes API. It is required for HPA to access CPU utilization metrics.

   You can install Metrics Server using the following commands:

```bash
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml
```

Verify that Metrics Server is running:

```bash
kubectl get deployment metrics-server -n kube-system
```

Ensure that the deployment status shows as running before proceeding with HPA setup.

HPA Manifest (nginx-hpa.yaml)

nginx-hpa.yaml

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50  # Scale up when CPU utilization exceeds 50%
```

kubectl Commands:

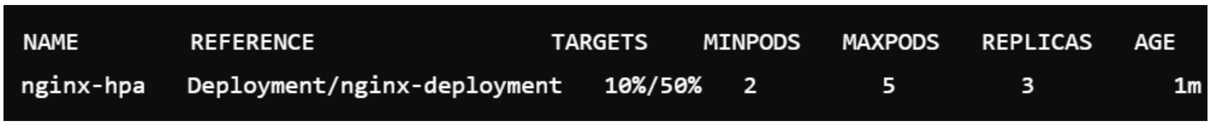
1. **Create the Horizontal Pod Autoscaler**:

```bash
kubectl apply -f nginx-hpa.yaml
```

2. **Validate HPA Creation**:

```bash
kubectl get hpa
```

This will show the status of the HPA:



```
NAME        REFERENCE                     TARGETS    MINPODS    MAXPODS    REPLICAS    AGE
nginx-hpa   Deployment/nginx-deployment     10%/50%    2          5          3           1m
```

3. **Simulate Load to Trigger Scaling**: To test auto-scaling, simulate CPU load on the Nginx Pods, and the HPA will increase the number of Pods automatically based on CPU utilization.

- First, open a shell into one of the running Nginx Pods using:

```bash
kubectl exec -it <nginx-pod-name> -- /bin/sh
```

- To simulate high CPU usage, run the following infinite loop inside the pod:

```bash
while true; do :; done
```

This loop will continuously keep the CPU busy, causing a CPU utilization spike. As the CPU load increases, the Horizontal Pod Autoscaler (HPA) will scale up the number of replicas if it exceeds the target CPU utilization.

4. **Monitor scaling:** You can monitor the HPA and pods to see the scaling in action using the following commands:

   - Check HPA metrics:

```bash
kubectl get hpa
```

This shows the current CPU utilization, the minimum and maximum replicas, and the current number of replicas.

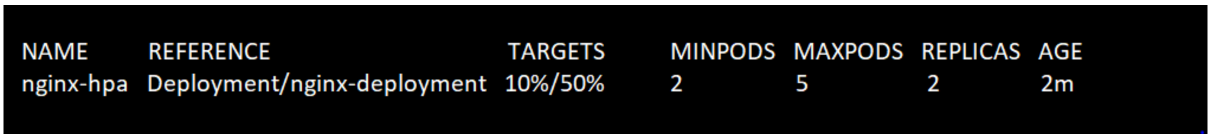
   - Watch the pod creation:

```bash
kubectl get pods -w
```

This command watches the pods in real time and will show new pods being created as the HPA scales up.

Sample Output

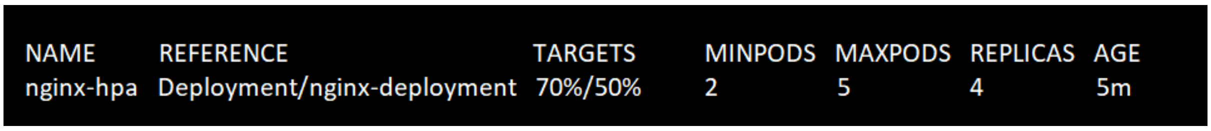
1. **Before Simulating Load (kubectl get hpa)**:



```
NAME       REFERENCE                     TARGETS    MINPODS  MAXPODS  REPLICAS  AGE
nginx-hpa  Deployment/nginx-deployment  10%/50%    2        5        2         2m
```

In this state, the CPU usage is low (10%), and there are only 2 replicas running (the initial setup).

2. **After Simulating Load (kubectl get hpa)**: After the load simulation, the CPU usage spikes, and the HPA begins scaling up:



```
NAME       REFERENCE                     TARGETS    MINPODS  MAXPODS  REPLICAS  AGE
nginx-hpa  Deployment/nginx-deployment  70%/50%    2        5        4         5m
```

3. **Watching Pods Scale Up (kubectl get pods -w)**:

```
NAME                              READY   STATUS    RESTARTS   AGE
nginx-deployment-6d97d589d8-abcde   1/1     Running   0          3m
nginx-deployment-6d97d589d8-fghij   1/1     Running   0          3m
nginx-deployment-6d97d589d8-klmno   1/1     Running   0          30s
nginx-deployment-6d97d589d8-pqrst   1/1     Running   0          10s
```

You can see new pods being created as the number of replicas increases. The new pods will be distributed across available nodes based on the cluster's scheduler.

4. **After Load Decreases**: Once the load stops (e.g., when you exit the infinite loop), the HPA will gradually scale down the number of replicas to the minimum defined (2 in this case). You'll see the pods terminating:

```
NAME                              READY   STATUS        RESTARTS   AGE
nginx-deployment-6d97d589d8-abcde   1/1     Running       0          10m
nginx-deployment-6d97d589d8-fghij   1/1     Running       0          10m
nginx-deployment-6d97d589d8-klmno   0/1     Terminating   0          7m
```

Summary of Results:

- **Initial state**: 2 replicas running, low CPU utilization.
- **After load simulation**: CPU utilization rises above the target (e.g., 70%/50%), causing the HPA to scale up, increasing the number of replicas.
- **Scaling up**: New pods are created to handle the load.
- **Scaling down**: Once the load decreases, the HPA scales the replicas back down to the minimum number.

This demonstrates how the HPA dynamically adjusts the number of pods based on real-time CPU usage.

Note:
In addition to scaling based on CPU utilization, the Horizontal Pod Autoscaler (HPA) can also scale pods based on other metrics depending on your application needs, such as:

- **Memory utilization**
- **Custom application metrics** (e.g., request rate, queue length, or response time)

🛠️ Step 6: Self-Healing with Kubernetes

Self-healing is a key feature of Kubernetes that ensures your application remains available by automatically replacing failed Pods. Here's how to observe and test the self-healing capabilities:

Deploy a Fault-Tolerant Application

We'll use the existing nginx-deployment to demonstrate self-healing. This Deployment already manages multiple replicas of NGINX Pods.

1. **Create the Deployment (if not already created):**

bash
```
kubectl apply -f nginx-deployment.yaml
```

2. **Check the Running Pods:**

bash
```
kubectl get pods
```

You should see a list of Pods managed by the nginx-deployment.

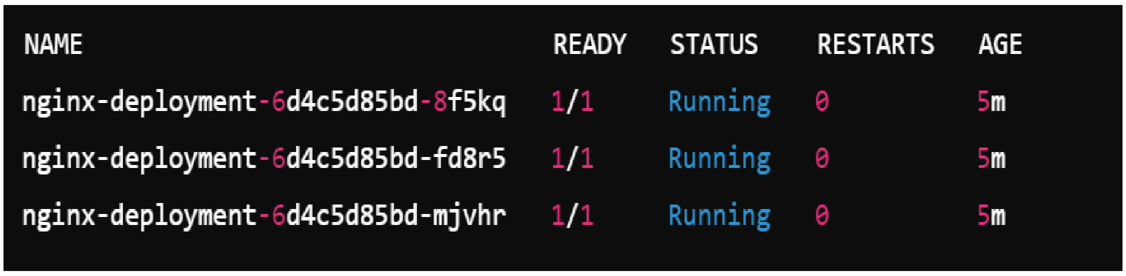
Simulate a Pod Failure

To observe self-healing, simulate a failure by deleting one of the Pods managed by the Deployment:

1. **Identify the Pod to Delete:**

bash
```
kubectl get pods
```

Example Output:



```
NAME                              READY   STATUS    RESTARTS   AGE
nginx-deployment-6d4c5d85bd-8f5kq  1/1     Running   0          5m
nginx-deployment-6d4c5d85bd-fd8r5  1/1     Running   0          5m
nginx-deployment-6d4c5d85bd-mjvhr  1/1     Running   0          5m
```

2. **Delete a Pod:**

bash
```
kubectl delete pod <nginx-pod-name>b
```

Replace <nginx-pod-name> with the name of one of the Pods listed.

Observe Self-Healing

Kubernetes will automatically detect the deleted Pod and create a new one to maintain the desired number of replicas:

1. **Monitor Pod Creation:**

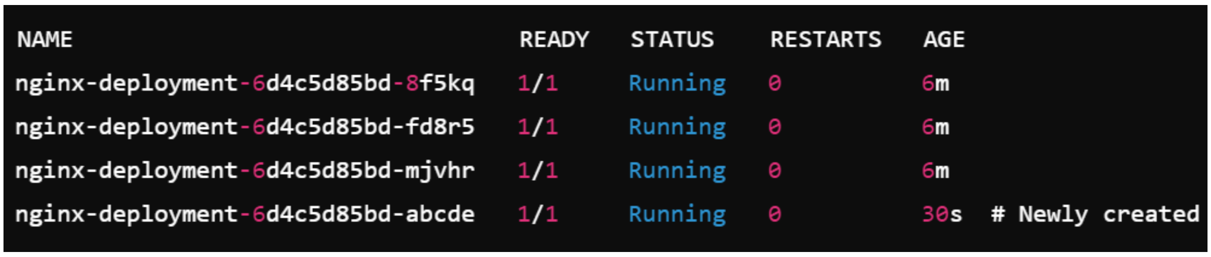
```bash
kubectl get pods -w
```

You will see Kubernetes creating a new Pod to replace the deleted one. The -w flag watches the Pods in real-time.

2. **Verify the New Pod:**

```bash
kubectl get pods
```

Example Output:



```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6d4c5d85bd-8f5kq   1/1     Running   0          6m
nginx-deployment-6d4c5d85bd-fd8r5   1/1     Running   0          6m
nginx-deployment-6d4c5d85bd-mjvhr   1/1     Running   0          6m
nginx-deployment-6d4c5d85bd-abcde   1/1     Running   0          30s  # Newly created
```

The new Pod (nginx-deployment-6d4c5d85bd-abcde in this example) will be created to replace the deleted Pod, ensuring that the total number of replicas matches the desired count (3).

🛠 Step 7: Cleanup

To remove the created resources (Pod, Deployment, and HPA), use the following kubectl commands:

1. **Delete the Pod**:

```bash
kubectl delete pod nginx-pod
```

2. **Delete the Deployment**:

```bash
kubectl delete deployment nginx-deployment
```

3. **Delete the Horizontal Pod Autoscaler (HPA):**

```bash
kubectl delete hpa nginx-hpa
```

✅ Summary

In this guide, we covered:

- How to create and manage Kubernetes **Pods** and **Deployments** with YAML manifest files.
- How to perform **rolling updates** to ensure zero downtime.
- Setting up **auto-scaling** using a **Horizontal Pod Autoscaler**.
- The necessary kubectl commands to create, validate, and manage these resources.

By leveraging **Deployments** and **HPA**, you can make your application more resilient, scalable, and manageable in Kubernetes. 🚀

📘 Q&A

Q1: What is the main purpose of a Kubernetes Deployment?

A Kubernetes Deployment automates the process of managing multiple replicas of a Pod, including tasks like rolling updates, auto-healing, and scaling.

Q2: How do I create a Pod with a YAML manifest?

To create a Pod, write a YAML file describing the Pod (e.g., nginx-pod.yaml) and use the command:

```bash
kubectl apply -f nginx-pod.yaml
```

Q3: What is the difference between a Pod and a Deployment in Kubernetes?

A **Pod** represents a single instance of a running process, while a **Deployment** manages multiple Pod replicas, ensures desired state, handles rolling updates, and supports auto-scaling.

Q4: How do I update an image version in a Kubernetes Deployment?

Modify the Deployment YAML to specify the new image version, then run:

```bash
kubectl apply -f <deployment-file.yaml>
```

Q5: What does kubectl rollout status do?

This command monitors the progress of a rolling update, allowing you to check if the update has been successfully deployed or if there are issues.

Q6: How do I enable auto-scaling for my Kubernetes Deployment?

Create a **Horizontal Pod Autoscaler (HPA)** YAML file defining the min/max number of Pods and CPU utilization target, then apply it with:

```bash
kubectl apply -f <hpa-file.yaml>
```

Q7: How can I simulate CPU load to test auto-scaling?

You can use tools like stress or ab (ApacheBench) to simulate CPU load on the Pods, which will trigger the Horizontal Pod Autoscaler to scale the Deployment.

Q8: How do I delete a Deployment in Kubernetes?

To delete a Deployment, use the command:

```bash
kubectl delete deployment <deployment-name>
```

Q9: What is Horizontal Pod Autoscaler (HPA) and how does it work?
The Horizontal Pod Autoscaler (HPA) automatically adjusts the number of replicas in a deployment based on observed CPU utilization (or other select metrics). It works by periodically checking the metrics of your Pods and scaling the number of replicas up or down to maintain the target CPU utilization or other metric values.

Q10: Why am I seeing <unknown> in the HPA TARGETS column?
<unknown> typically means that the HPA is unable to retrieve CPU metrics, often due to issues with the Metrics Server. Ensure that the Metrics Server is correctly installed and running, and verify that your Pods are reporting metrics correctly.

Q11: How do I simulate load to test HPA?
To simulate load, you can use a simple infinite loop inside a Pod to generate CPU usage. For example:

```bash
while true; do :; done
```

This loop will keep the CPU busy, causing the HPA to scale up the number of Pods if the CPU utilization exceeds the target threshold.

Q12: How can I monitor the scaling activity of the HPA?
Use the following commands to monitor scaling:

- Check HPA status: kubectl get hpa
- Watch Pods: kubectl get pods -w These commands will show you the current number of replicas, CPU utilization, and real-time changes in the number of Pods.

Q13: What should I be cautious about when simulating load?
Be aware that simulating high CPU load can impact other applications running on the same cluster. Ensure that you perform load tests in a controlled environment, such as a development or test cluster, and stop the load simulation once testing is complete to avoid unnecessary resource consumption.

Q14: How do I stop the CPU load simulation?
To stop the load simulation, you can exit the infinite loop by pressing Ctrl + C in the kubectl exec shell, or terminate the Pod running the load simulation if you used a separate Pod for testing.

Q15: Why is my HPA not scaling up despite high CPU utilization?
If the HPA is not scaling up, check the following:

- Ensure that the CPU requests are defined in your deployment.
- Verify that the Metrics Server is properly installed and functioning.
- Review the HPA configuration to confirm that the target CPU utilization is set correctly.

Q16: Can I scale my application based on metrics other than CPU?
A: Yes, you can scale based on other metrics such as **memory utilization**, **custom application metrics** (like request rate, queue length, or response time), and many others.

Q17: Can I scale based on multiple metrics at the same time?
A: Yes, you can configure the Horizontal Pod Autoscaler (HPA) to scale based on multiple metrics, including CPU, memory, and custom metrics simultaneously, depending on your application needs. You just need to define the different metrics in the HPA configuration.