**Study Material: Creating and Scaling Deployments in Kubernetes**

**Introduction**

In Kubernetes, a **Deployment** is a higher-level abstraction that manages **Pods** and allows you to scale, update, and manage your application easily. Unlike creating individual Pods, Deployments enable you to manage multiple replicas of your application and distribute the load across your cluster.

**Step 1: Create a Deployment**

Let’s create a Deployment that manages multiple Pods running the **Nginx** web server.

**Create an Nginx Deployment:**

1. Use the kubectl create deployment command to create a Deployment:
2. kubectl create deployment nginx-deployment --image=nginx
   * nginx-deployment: Name of the Deployment.
   * --image=nginx: Specifies the Docker image to use (in this case, nginx).
3. Verify the Deployment:
4. kubectl get deployments

You’ll see the nginx-deployment with **1/1** replicas ready.

1. List the Pods managed by the Deployment:
2. kubectl get pods

You’ll see a single Pod with a name like nginx-deployment-<hash>.

**Step 2: Explore the Deployment Details**

Let’s explore the details of the Deployment to understand how it manages Pods.

**Describe the Deployment:**

1. Describe the Deployment to see detailed information:
2. kubectl describe deployment nginx-deployment

You’ll see details like:

* + **Namespace**: default
  + **Labels**: Automatically assigned labels (e.g., app=nginx-deployment).
  + **Replicas**: Desired and current number of Pods.
  + **Selector**: Used to match Pods to the Deployment.
  + **Strategy**: Rolling update strategy for updates.

1. Look for the **ReplicaSet** section. A ReplicaSet is responsible for managing the Pods in the Deployment.

**Describe the Pod:**

1. Get the name of the Pod:
2. kubectl get pods
3. Describe the Pod to see its details:
4. kubectl describe pod <pod-name>

You’ll see:

* + **Node**: The node where the Pod is running.
  + **IP Address**: The internal IP address of the Pod.
  + **Labels**: Matches the selector in the Deployment.
  + **Containers**: The container(s) running inside the Pod.

**Step 3: Scale the Deployment**

One of the key benefits of using a Deployment is the ability to scale the number of Pods up or down.

**Scale the Deployment to 5 Replicas:**

1. Scale the Deployment to 5 replicas:
2. kubectl scale deployment nginx-deployment --replicas=5
3. Verify the scaling:
4. kubectl get pods

You’ll see 5 Pods with names like nginx-deployment-<hash>.

1. Check the Deployment status:
2. kubectl get deployments

You’ll see **5/5** replicas ready.

**Scale the Deployment Down to 3 Replicas:**

1. Scale the Deployment down to 3 replicas:
2. kubectl scale deployment nginx-deployment --replicas=3
3. Verify the scaling:
4. kubectl get pods

You’ll see only 3 Pods running.

**Step 4: Access the Pods**

By default, Pods have internal IP addresses that are not accessible from outside the cluster. To access them, you need to create a **Service**. However, you can test connectivity from within the cluster.

**Connect to a Pod from Inside the Cluster:**

1. SSH into the Minikube node:
2. minikube ssh
3. Get the IP address of one of the Pods:
4. kubectl get pods -o wide
5. Use curl to connect to the Pod’s IP address:
6. curl <pod-ip>

You’ll see the Nginx welcome page.

1. Exit the SSH session:
2. exit

**Step 5: Clean Up**

Let’s delete the Deployment to clean up.

**Delete the Deployment:**

1. Delete the Deployment:
2. kubectl delete deployment nginx-deployment
3. Verify that the Deployment and Pods are deleted:
4. kubectl get deployments
5. kubectl get pods

You’ll see: **No resources found in default namespace.**

**Key Takeaways**

* **Deployments** manage Pods and allow you to scale, update, and roll back your application.
* Use kubectl create deployment to create a Deployment.
* Use kubectl scale to scale the number of Pods in a Deployment.
* Pods have internal IP addresses and are not accessible from outside the cluster by default.
* Use **Services** to expose Pods to the outside world (covered in the next section).

**Commands Summary**

| **Command** | **Description** |
| --- | --- |
| kubectl create deployment <name> --image=<img> | Create a Deployment. |
| kubectl get deployments | List all Deployments. |
| kubectl describe deployment <name> | Show detailed information about a Deployment. |
| kubectl scale deployment <name> --replicas=<n> | Scale a Deployment to n replicas. |
| kubectl delete deployment <name> | Delete a Deployment. |

**Next Steps**

Now that you’ve created and scaled a Deployment, you can:

* Expose the Deployment to the outside world using a **Service**.
* Perform rolling updates and rollbacks.
* Explore advanced Deployment strategies.

With this knowledge, you’re ready to manage scalable applications in Kubernetes! 🚀

### Study Material: Creating and Scaling Deployments in Kubernetes

---

#### \*\*Introduction\*\*

In Kubernetes, a \*\*Deployment\*\* is a higher-level abstraction that manages \*\*Pods\*\* and allows you to scale, update, and manage your application easily. Unlike creating individual Pods, Deployments enable you to manage multiple replicas of your application and distribute the load across your cluster.

---

### \*\*Step 1: Create a Deployment\*\*

Let’s create a Deployment that manages multiple Pods running the \*\*Nginx\*\* web server.

---

#### \*\*Create an Nginx Deployment:\*\*

1. Use the `kubectl create deployment` command to create a Deployment:

```bash

kubectl create deployment nginx-deployment --image=nginx

```

- `nginx-deployment`: Name of the Deployment.

- `--image=nginx`: Specifies the Docker image to use (in this case, `nginx`).

2. Verify the Deployment:

```bash

kubectl get deployments

```

You’ll see the `nginx-deployment` with \*\*1/1\*\* replicas ready.

3. List the Pods managed by the Deployment:

```bash

kubectl get pods

```

You’ll see a single Pod with a name like `nginx-deployment-<hash>`.

---

### \*\*Step 2: Explore the Deployment Details\*\*

Let’s explore the details of the Deployment to understand how it manages Pods.

---

#### \*\*Describe the Deployment:\*\*

1. Describe the Deployment to see detailed information:

```bash

kubectl describe deployment nginx-deployment

```

You’ll see details like:

- \*\*Namespace\*\*: `default`

- \*\*Labels\*\*: Automatically assigned labels (e.g., `app=nginx-deployment`).

- \*\*Replicas\*\*: Desired and current number of Pods.

- \*\*Selector\*\*: Used to match Pods to the Deployment.

- \*\*Strategy\*\*: Rolling update strategy for updates.

2. Look for the \*\*ReplicaSet\*\* section. A ReplicaSet is responsible for managing the Pods in the Deployment.

---

#### \*\*Describe the Pod:\*\*

1. Get the name of the Pod:

```bash

kubectl get pods

```

2. Describe the Pod to see its details:

```bash

kubectl describe pod <pod-name>

```

You’ll see:

- \*\*Node\*\*: The node where the Pod is running.

- \*\*IP Address\*\*: The internal IP address of the Pod.

- \*\*Labels\*\*: Matches the selector in the Deployment.

- \*\*Containers\*\*: The container(s) running inside the Pod.

---

### \*\*Step 3: Scale the Deployment\*\*

One of the key benefits of using a Deployment is the ability to scale the number of Pods up or down.

---

#### \*\*Scale the Deployment to 5 Replicas:\*\*

1. Scale the Deployment to 5 replicas:

```bash

kubectl scale deployment nginx-deployment --replicas=5

```

2. Verify the scaling:

```bash

kubectl get pods

```

You’ll see 5 Pods with names like `nginx-deployment-<hash>`.

3. Check the Deployment status:

```bash

kubectl get deployments

```

You’ll see \*\*5/5\*\* replicas ready.

---

#### \*\*Scale the Deployment Down to 3 Replicas:\*\*

1. Scale the Deployment down to 3 replicas:

```bash

kubectl scale deployment nginx-deployment --replicas=3

```

2. Verify the scaling:

```bash

kubectl get pods

```

You’ll see only 3 Pods running.

---

### \*\*Step 4: Access the Pods\*\*

By default, Pods have internal IP addresses that are not accessible from outside the cluster. To access them, you need to create a \*\*Service\*\*. However, you can test connectivity from within the cluster.

---

#### \*\*Connect to a Pod from Inside the Cluster:\*\*

1. SSH into the Minikube node:

```bash

minikube ssh

```

2. Get the IP address of one of the Pods:

```bash

kubectl get pods -o wide

```

3. Use `curl` to connect to the Pod’s IP address:

```bash

curl <pod-ip>

```

You’ll see the Nginx welcome page.

4. Exit the SSH session:

```bash

exit

```

---

### \*\*Step 5: Clean Up\*\*

Let’s delete the Deployment to clean up.

---

#### \*\*Delete the Deployment:\*\*

1. Delete the Deployment:

```bash

kubectl delete deployment nginx-deployment

```

2. Verify that the Deployment and Pods are deleted:

```bash

kubectl get deployments

kubectl get pods

```

You’ll see: \*\*No resources found in default namespace.\*\*

---

### \*\*Key Takeaways\*\*

- \*\*Deployments\*\* manage Pods and allow you to scale, update, and roll back your application.

- Use `kubectl create deployment` to create a Deployment.

- Use `kubectl scale` to scale the number of Pods in a Deployment.

- Pods have internal IP addresses and are not accessible from outside the cluster by default.

- Use \*\*Services\*\* to expose Pods to the outside world (covered in the next section).

---

### \*\*Commands Summary\*\*

| \*\*Command\*\* | \*\*Description\*\* |

|--------------------------------------------------|------------------------------------------------------|

| `kubectl create deployment <name> --image=<img>` | Create a Deployment. |

| `kubectl get deployments` | List all Deployments. |

| `kubectl describe deployment <name>` | Show detailed information about a Deployment. |

| `kubectl scale deployment <name> --replicas=<n>` | Scale a Deployment to `n` replicas. |

| `kubectl delete deployment <name>` | Delete a Deployment. |

---

### \*\*Next Steps\*\*

Now that you’ve created and scaled a Deployment, you can:

- Expose the Deployment to the outside world using a \*\*Service\*\*.

- Perform rolling updates and rollbacks.

- Explore advanced Deployment strategies.

---

With this knowledge, you’re ready to manage scalable applications in Kubernetes! 🚀