**[k8s Made Easy] 2**

Deployment

A **Deployment** in Kubernetes is like a **manager** for your application Pods. It handles the creation, updating, and scaling of Pods automatically. This makes it easier for you to manage your applications without dealing with individual Pods manually.

**Key Features of a Deployment:**

**Declarative Updates:** You tell Kubernetes what you want (e.g., 3 instances of an app), and it takes care of making it happen.

**Rolling Updates:** Allows smooth updates without downtime by gradually replacing old Pods with new ones.

**Rollback:** If something goes wrong during an update, you can easily go back to a previous version.

**Scaling:** Quickly increase or decrease the number of Pods as needed.

**Example:**

If you have a web application with a Deployment managing 3 Pods, and one Pod crashes, the Deployment will automatically create a new Pod to replace the failed one.

# Understanding Pods

A **Pod** is the smallest unit in Kubernetes. It represents a single instance of your running application, usually containing **one or more containers** (e.g., Docker containers).

Think of a Pod as a wrapper around your application container, along with its storage and network settings.

**Example:**

If your app is a web server running inside a Docker container, Kubernetes will place that container inside a Pod.

# Differences Between a Pod and a Deployment

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Pod** | **Deployment** |
| **Abstraction** | Smallest unit, usually holds one or more containers. | Manages a group of identical Pods automatically. |
| **Management** | Created and deleted manually; not self-healing. | Handles Pods for you, replacing them if they fail. |
| **Updates** | Changes require manual intervention, causing downtime. | Provides rolling updates without downtime. |
| **Scaling** | Need to manually add or remove Pods. | Easily scale up or down with one command. |
| **Self-Healing** | Needs a controller to restart if it crashes. | Automatically replaces failed Pods. |

**Example Scenario:**

Imagine you are running a web app for an online store:

 **With just Pods:** If one of your Pods crashes, you need to restart it manually, and users might face downtime.

 **With a Deployment:** The Deployment will automatically detect the crash, replace the Pod, and keep your app running smoothly.

**Kubernetes ReplicaSet Simplified Explanation)**

A **ReplicaSet** is a controller in Kubernetes that makes sure the **right number of Pods** are always running. If a Pod crashes or is accidentally deleted, the ReplicaSet will automatically replace it to maintain the desired number.

**Key Points About ReplicaSets:**

**Maintains Stability:** Ensures that a specified number of Pod replicas are always running.

**Automatic Recovery:** If any Pod goes down, the ReplicaSet will detect this and create a new Pod to replace it.

**Used by Deployments:** When you create a Deployment, it automatically uses a ReplicaSet to manage the Pods.

**Example:**

You want 3 instances of your web server running. The ReplicaSet checks constantly to ensure there are always exactly 3 Pods. If one Pod crashes, the ReplicaSet quickly spins up a new one.

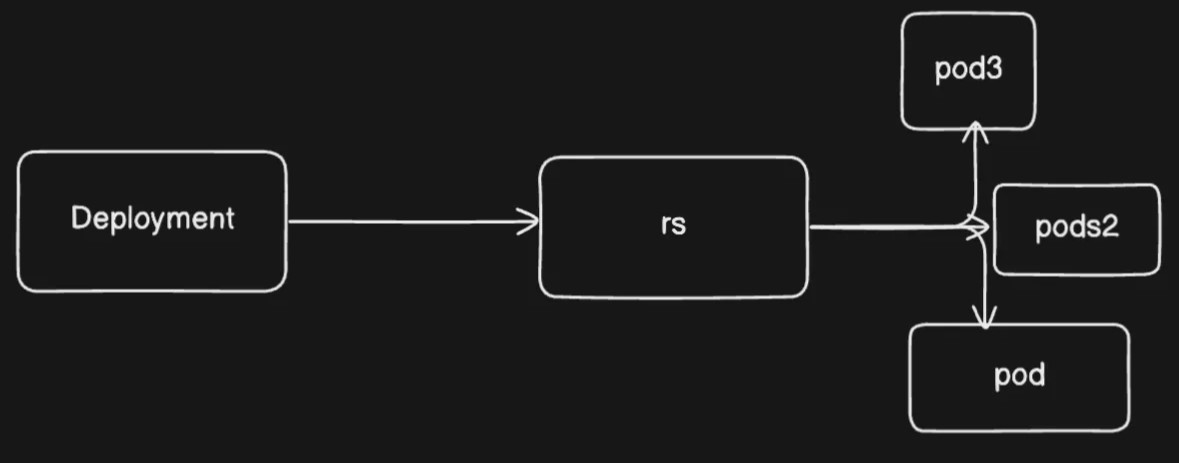
# How It Works: Series of Events

 **User creates a Deployment:** This defines the number of replicas needed (e.g., 3 replicas of a web server).

 **Deployment creates a ReplicaSet:** The ReplicaSet is responsible for maintaining the specified number of Pods.

 **ReplicaSet creates the Pods:** It ensures there are always the correct number of Pods running.

 **If a Pod goes down:** The ReplicaSet notices it and automatically brings up a new Pod to replace it.



**Example Scenario:**

You have a Node.js app running with a Deployment specifying 3 replicas. If one of the Pods fails, the ReplicaSet controller detects it and creates a new Pod, so your app always has 3 instances running.

**Simplified Visual Flow:**

 **Deployment  ReplicaSet  Pods**

 **Pod Failure Detected  ReplicaSet Creates New Pod**

This flow ensures high availability and reliability for your applications.

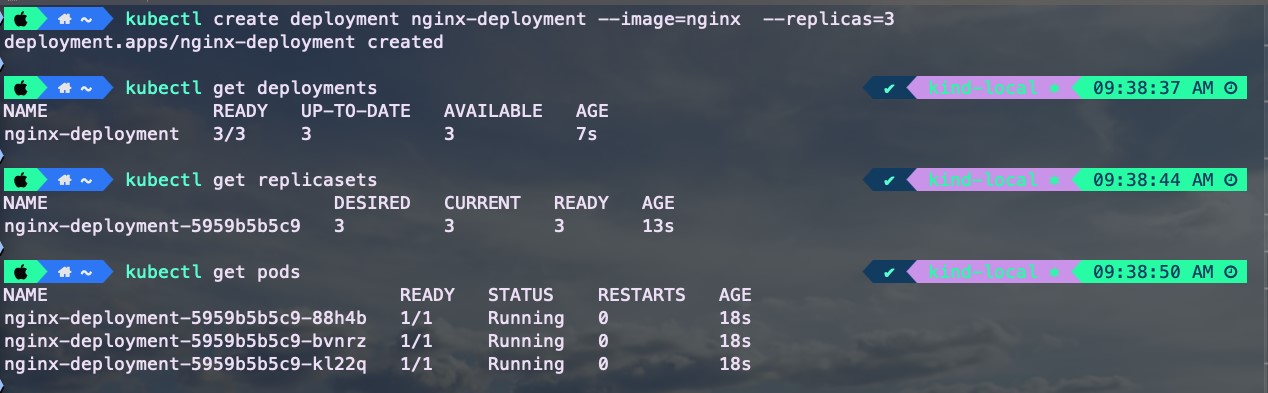
Series of Events:

# Step-by-Step Breakdown of Creating a Deployment in Kubernetes

When you run this command:

kubectl create deployment nginx-deployment --image=nginx --re

plicas=3



A lot happens behind the scenes in your Kubernetes cluster. Letʼs break it down step by step:

# 1. Command Execution

You run the command using **kubectl** (the command-line tool for Kubernetes). Your kubectl is already connected to your Kubernetes cluster.

# 2. API Request

The kubectl command sends a request to the **Kubernetes API server**, asking it to create a new Deployment with the given parameters (image: nginx, replicas: 3.

# 3. API Server Processing

The **API server** receives this request, checks if itʼs valid, and updates the clusterʼs desired state with the new Deployment configuration.

# 4. Storage in etcd

The **desired state** of the Deployment is saved in **etcd**, which is a special database used by Kubernetes to store all its configuration and state information. Think of etcd as the **brain** of Kubernetes that keeps track of everything.

# 5. Deployment Controller Monitoring

The **Deployment controller** (part of the Kubernetes control plane) constantly monitors for new or updated Deployments. It detects the Deployment you just created.

# 6. ReplicaSet Creation

The Deployment controller creates a **ReplicaSet**, which is responsible for maintaining the specified number of Pods 3 in this case). The ReplicaSet acts like a **safety manager**, making sure there are always 3 Pods running.

# 7. Pod Creation

The **ReplicaSet controller** now ensures that 3 Pods are created based on the Deploymentʼs instructions. It asks the API server to create these Pods.

# 8. Scheduler Assignment

The **Kubernetes scheduler** looks for new Pods that are in a "Pending" state (waiting to be assigned to a node). It decides the best node for each Pod based on the available resources.

# 9. Node and Kubelet

Once the Pods are assigned to nodes, the **kubelet** (an agent running on each node) pulls the **nginx container image** and starts the containers as specified in the Pod definition.

# Summary: Hierarchical Flow

 **User** → **kubectl** → **API Server** → **etcd (storage)**

 **Deployment Controller** → **ReplicaSet** → **Pod Creation**

 **Scheduler** → **Node Assignment** → **kubelet Starts Containers**

**Common Student Question:**

**"Why use a Deployment when a ReplicaSet can already handle Pods?"**

Good question! While a **ReplicaSet** can manage Pods and handle failures, a **Deployment** gives you **more control** and features:

**Rolling Updates:** Update your application smoothly without downtime.

**Rollback:** Easily revert to a previous version if something goes wrong.

**Declarative Updates:** You only need to tell Kubernetes the desired state, and it takes care of the rest.

In short, a Deployment is a **higher-level manager** that uses a ReplicaSet for extra capabilities, making your app management easier and safer.

**Kubernetes Hierarchical Relationship**

In Kubernetes, there is a clear hierarchy that helps manage your applications efficiently. Letʼs break it down from the top-level controller to the smallest unit.

# 1. Deployment: High-Level Manager

A **Deployment** is like a **project manager** for your application. It handles the entire lifecycle of your app, including:

**Creating and Updating ReplicaSets:** When you create or update a Deployment, it sets up or changes the underlying ReplicaSets.

**Rolling Updates and Rollbacks:** It manages smooth updates (rolling updates) without downtime and can easily roll back to a previous version if needed.

**Scaling:** You can easily scale your application up or down by changing the number of replicas in the Deployment.

**Example:** If you want to update your web app to a new version, the Deployment will create a new ReplicaSet for the updated version and gradually replace the old Pods with new ones.

# 2. ReplicaSet: Mid-Level Manager

A **ReplicaSet** acts like a **team leader** that ensures the desired number of Pods are always running. Its main roles are:

**Maintaining Desired State:** The ReplicaSet monitors the number of Pods and makes sure the correct number (e.g., 3 replicas) are running at all times.

**Automatic Pod Management:** If a Pod crashes or is deleted, the ReplicaSet creates a new Pod to replace it.

**Label Selector:** Uses labels (like tags) to identify and manage the Pods it controls.

**Example:** If you specify 5 replicas of a web server, the ReplicaSet keeps an eye on this and automatically creates new Pods if one goes down.

# 3. Pod: Lowest-Level Unit

A **Pod** is the smallest building block in Kubernetes. Itʼs like a **single worker** that runs one or more containers (e.g., Docker containers).

**Basic Unit of Deployment:** A Pod represents a single instance of your running application.

**Contains One or More Containers:** It usually runs a single container, but can run multiple if they are tightly coupled (e.g., a web server and a logging sidecar).

**Example:** If you have an nginx web server container, it will be placed inside a Pod, which handles its networking and storage settings.

**Simplified Hierarchy Overview:**

 **Deployment:** The project manager — handles the big picture (updates, scaling, rollbacks).

 **ReplicaSet:** The team leader — maintains the right number of Pods.

 **Pod:** The worker — runs the actual application (container).

**Analogy for Better Understanding:**

Imagine a **restaurant kitchen**:

The **Deployment** is like the head chef who plans the menu, manages staff, and makes decisions on changes (updates).

The **ReplicaSet** is like the kitchen manager who ensures there are always enough chefs Pods) cooking the dishes.

The **Pod** is a chef who prepares and cooks the actual dish (container).

This simplified explanation should help your students easily understand the hierarchical relationship between these components. Let me know if you want more analogies or examples!

# Creating a ReplicaSet in Kubernetes

In this section, we will **focus only on ReplicaSets** (without using a Deployment). We will create a ReplicaSet that manages **3 Pods** running the **nginx** container.

# Step 1: Define the ReplicaSet (rs.yml)

Create a YAML file named **rs.yml** with the following content:

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-replicaset

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

-

name: nginx

image: nginx:latest

ports:

-

containerPort:

80

**Explanation:**

**apiVersion:** The version of the API to use (apps/v1. **kind:** Specifies that we are creating a **ReplicaSet**.

**metadata:** Contains the name of the ReplicaSet ( nginx-replicaset ).

**replicas:** The number of Pods to run 2 replicas). **selector:** Identifies the Pods managed by this ReplicaSet using labels ( app:

nginx ).

**template:** Defines the Pod template with its labels, container details (name, image, and exposed port).

# Step 2: Apply the Manifest

Use the following command to create the ReplicaSet:

kubectl apply -f rs.yml

# Step 3: Check the ReplicaSet Status

Run this command to see the details of your ReplicaSet:

kubectl get rs

**Output:**

NAME DESIRED CURRENT READY AGE

nginx-replicaset 2 2 2 9s

**DESIRED**

Number of Pods specified 2.

**CURRENT**

Number of Pods currently running.

**READY**

Number of Pods that are up and ready.

# Step 4: Check the Pods

Use this command to list the Pods created by the ReplicaSet:

kubectl get pods

**Example Output:**

NAME READY STATUS RESTARTS AGE nginx-replicaset-7h4rt 1/1 Running 0 17s nginx-replicaset-l4pdq 1/1 Running 0 17s

The Pods are named after the ReplicaSet ( nginx-replicaset ) followed by a **unique identifier** (e.g., 7h4rt ).

# Step 5: Test the Self-Healing Feature

Try deleting one of the Pods using the command below:

kubectl delete pod nginx-replicaset-7h4rt

Then, check the Pods again:

kubectl get pods

**What happens?**

The deleted Pod will be **recreated automatically** by the ReplicaSet to maintain the desired state of 2 replicas.

# Step 6: Test Adding an Extra Pod

Letʼs try creating an additional Pod with the same label ( app=nginx ):

kubectl run nginx-pod --image=nginx --labels="app=nginx"

**Expected Result:**

The extra Pod will be **terminated automatically** because the ReplicaSet already has the desired number of 2 Pods.

**Step 7: Clean Up the ReplicaSet** Delete the ReplicaSet when you are done:

kubectl delete rs nginx-replicaset

💡 **Tip:**

Notice the naming pattern of the Pods: they follow the format **<replicaset-name><unique-id>** (e.g., nginx-replicaset-7h4rt ).

This version provides a step-by-step guide with clear explanations, making it easier for students to understand and follow along. Let me know if you need further adjustments!

# Creating a Deployment in Kubernetes

A **Deployment** is a higher-level abstraction that manages the lifecycle of your application, ensuring that the desired number of Pods are running and allowing for updates and rollbacks.

In this section, we will create a Deployment that starts **3 Pods** running the **nginx** container.

# Step 1: Define the Deployment (deployment.yml)

Create a YAML file named **deployment.yml** with the following content:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

-

name: nginx

image: nginx:latest

ports:

-

containerPort:

80

**Explanation:**

**apiVersion:** The version of the API  apps/v1 ). **kind:** Specifies that we are creating a **Deployment**.

**metadata:** Contains the name of the Deployment ( nginx-deployment ). **replicas:** Number of Pods to create 2 replicas).

**selector:** Specifies the label ( app: nginx ) to identify the Pods managed by this Deployment.

**template:** Defines the Pod template with metadata (labels), container details (name, image), and the port to expose.

# Step 2: Apply the Deployment

Use the following command to create the Deployment:

kubectl apply -f deployment.yml

This command tells Kubernetes to create the Deployment based on the configuration in the **deployment.yml** file.

# Step 3: Verify the Deployment

Check the status of the Deployment with:

kubectl get deployment

**Example Output:**

NAME READY UP-TO-DATE AVAILABLE AGE nginx-deployment 2/2 2 2 8s

**READY** Shows how many Pods are up and running 2/2 means all Pods are running).

**UPTODATE** The number of Pods updated to the latest configuration.

**AVAILABLE** The number of Pods ready to serve requests.

# Step 4: Check the ReplicaSet

The Deployment creates a **ReplicaSet** automatically. To see it, run:

kubectl get rs

**Example Output:**

NAME DESIRED CURRENT READY AGE

nginx-deployment-54b9c68f67 2 2 2 71s

**DESIRED**

Number of Pods specified 2.

**CURRENT**

Number of Pods currently running.

**READY**

Number of Pods that are ready.

# Step 5: Check the Pods

To list the Pods created by the Deployment, use:

kubectl get pods

**Example Output:**

NAME READY STATUS RESTART S AGE nginx-deployment-54b9c68f67-smz8p 1/1 Running 0

114s nginx-deployment-54b9c68f67-srtqx 1/1 Running 0

114s

The Pods are named based on the Deployment name ( nginx-deployment ) followed by a **ReplicaSet identifier** and a **unique Pod ID**.

# Step 6: Test Self-Healing

Try deleting one of the Pods using:

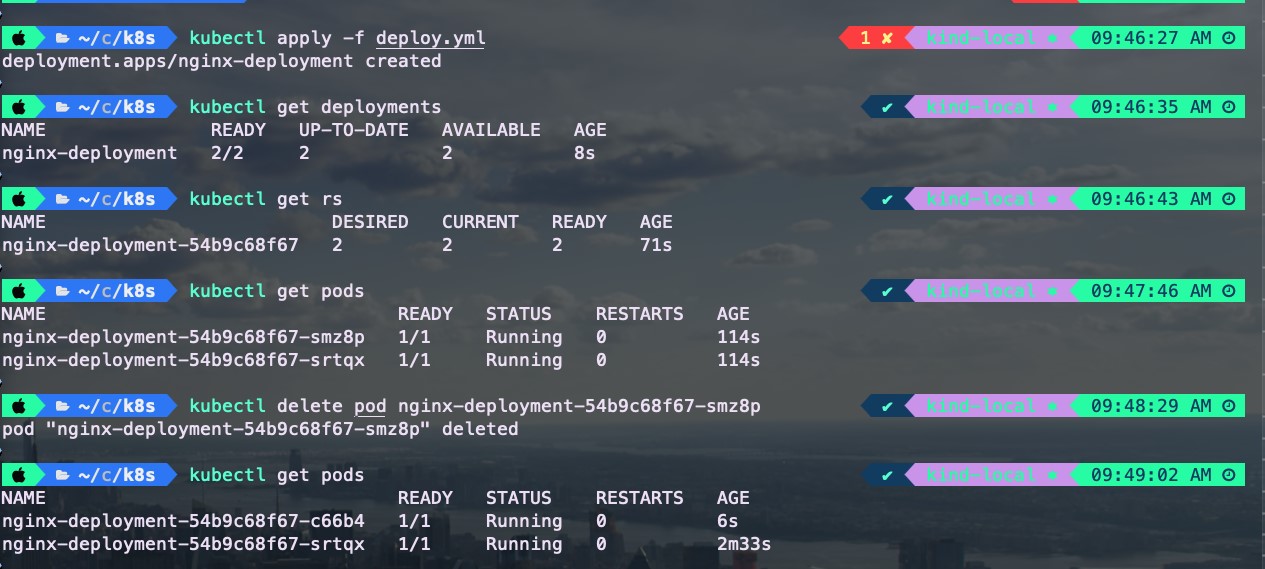
kubectl delete pod nginx-deployment-54b9c68f67-smz8p

Then, check the Pods again:

kubectl get pods

**What happens?**

The deleted Pod is **automatically recreated** by the Deployment to maintain the desired state of 2 Pods.



**Why Use Deployments Instead of Just ReplicaSets?**

**Ease of Management:** Deployments provide features like rolling updates, rollbacks, and versioning, making it easier to manage updates without downtime.

**Declarative Updates:** You can specify the desired state, and Kubernetes will ensure it is met automatically.

**Self-Healing:** If a Pod crashes or is deleted, the Deployment ensures it is replaced without manual intervention.

This explanation should help your students grasp the concepts better and follow along with the practical examples. Let me know if you need any more tweaks!