Deployment:

A Kubernetes deployment is a resource object in Kubernetes that provides declarative updates to applications. A deployment allows you to describe an application's life cycle, such as which images to use for the app, the number of pods there should be, and the way in which they should be updated.

When a deployment is created, it creates a replicaset of it's own and pods of the replicaset.

Key Features of Deployments:

Declarative Updates: Deployments allow you to specify how you want your application to run. The deployment controller handles the intricate details of creating, scaling, and updating pods to reach the desired state.

Rolling Updates: Deployments excel at performing rolling updates. They gradually introduce new pods with updated configurations while phasing out old ones. This minimizes downtime and ensures a smooth transition during application updates.

Rollback Capability: If an update introduces issues, deployments allow you to easily rollback to a previous stable version. This provides a safety net and simplifies rollback procedures.

Scaling: Scaling your application up or down becomes effortless with deployments. You simply adjust the desired number of replicas in the deployment definition, and the controller handles the scaling process.

Health Checks: Deployments can integrate with liveness and readiness probes to monitor the health of your pods. These probes determine if a pod is healthy and ready to receive traffic, ensuring only healthy pods serve requests.

Blue/Green Deployments: Deployments facilitate advanced deployment strategies like blue/green deployments. This approach allows you to create a new version of your application (green) alongside the existing one (blue) and then switch traffic over to the green deployment if everything checks out.

DEPLOYMENT DEFINITION FILE

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-dep
spec:
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web-app
        image: nginx
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
```

Commands:

- kubectl get deployments
- kubectl get deployment [deployment name] -o yaml > deployment.yaml
- kubectl apply -f deployment.yaml
- kubectl describe deployment deploymentName
- kubectl delete deployment deploymentName
- kubectl scale deployment deploymentName --replicas [number]
- kubectl logs deployment deploymentName
- kubectl logs deployment deploymentName -c containerName

Updating Pod commands

- kubectl rollout SUBCOMMAND
  in the place of SUBCOMMAND we can use status, pause, resume, restart, undo, history

- kubectl set image deployment deploymentName containerName newImage --record

OR

- we can edit the image spec in deployment-definition file and use “kuectl replace -f filename” to perform rolling update

OR

- we can also use “kubectl edit deployment deploymentName” to update the deployment

Difference between Deployment and ReplicaSet

Deployment	ReplicaSet
High level Abstraction that Manages ReplicaSet.	A lower level Abstraction that manages the desired number of replica of a pod.
It provides additional features such as rolling updates, rollbacks, and versioning of the application	Additionally it provides basic scaling and self healing mechanism to a pod.
It provides a mechanism for rolling update and rollback for the application enabling seamless updates and reducing downtime.	Application must be manually updated or rollback.
It provides versioning of the application, allowing us to manage multiple versions of our application. It also makes easy to roll back to previous version if necessary.	ReplicaSet doesn’t provide these features.