# What **dynamic provisioning using a StorageClass** means in Kubernetes ?

## What is Dynamic Provisioning?

### Static Provisioning (manual way):

Earlier, we had to **manually create** a PersistentVolume (PV) and then a PVC (PersistentVolumeClaim) would try to match it. Like:

> You pre-book a hotel room and wait for someone to come and ask for that specific room.

---

### Dynamic Provisioning (automatic way):

With **dynamic provisioning**, Kubernetes **automatically creates a PersistentVolume (PV)** when a user creates a PVC — **based on a StorageClass** definition.

> You walk into a hotel and say, "I want a room with AC and Wi-Fi", and the receptionist (StorageClass) assigns and sets up a new room just for you.

---

## What is a StorageClass?

A **StorageClass** is like a blueprint or template that tells Kubernetes **how to create storage dynamically**.

It includes:

- **Provisioner**: The plugin/driver used to create volumes (like AWS EBS, GCE PD, or local-path).
- **Parameters**: Special instructions for how to create the volume (type, size, speed, etc.).
- **Reclaim Policy**: What to do with the volume after it's released (retain/delete).

---

### 🛠 Example of a StorageClass

```
yaml
---------
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: rancher.io/local-path
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

When a user creates a PVC like this:

```
yaml
---------
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Kubernetes will:

1. **Look up the** `local-storage` **StorageClass.**
2. **Use the** `local-path-provisioner` **(provisioner plugin) to create a PV.**
3. **Bind that new PV to the PVC automatically.**

---

## Summary

| Term | Meaning |
|---|---|
| StorageClass | Template that defines *how* and *where* to create storage |
| Dynamic Provisioning | Automatically creates PVs based on user PVCs and StorageClass |
| Provisioner | Plugin/driver responsible for creating the actual volume |
| PVC | Request for storage from a user/app |

---

## Dynamic Provisioning with StorageClass on Kind Cluster

### Prerequisites

We'll use **Kind**, and for dynamic provisioning, we'll enable the **default storage class** using the built-in `hostPath` provisioner with the help of a simple local provisioner like `rancher/local-path-provisioner`, which works well with Kind.

---

---

## 1️⃣ Create Kind cluster with volume mounts

Let’s mount a directory from the host into the Kind node to simulate storage:

```
bash
---------
```

### 1. Create the config file (e.g., kind-config.yaml):

```
yaml

# kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraMounts:
      - hostPath: /tmp/dynamic-pv
        containerPath: /tmp/dynamic-pv
```

**2Run the `kind` command using the file:**

```
bash

kind create cluster --name dynamic-pv-demo --config kind-config.yaml
```

---

## 2️⃣ Install local-path-provisioner (acts like a dynamic PV controller)

```
bash
---------
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-
provisioner/master/deploy/local-path-storage.yaml
```

Make it the default storage class:

```
bash
---------
kubectl patch storageclass local-path -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

---

## 3️⃣ Create PVC (no need to define a PV manually)

```
yaml
---------
# pvc-dynamic.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Apply it:

```
bash
---------
kubectl apply -f pvc-dynamic.yaml
kubectl get pvc
```

You should see `STATUS: Bound` — this means a PV has been automatically created.

---

## 4️⃣ Create a Pod that uses this PVC

```
yaml
---------
# pod-dynamic.yaml
apiVersion: v1
kind: Pod
metadata:
  name: dynamic-pv-pod
spec:
  containers:
    - name: my-container
      image: busybox
      command: ["sh", "-c", "sleep 3600"]
      volumeMounts:
        - mountPath: "/data"
          name: storage
  volumes:
    - name: storage
      persistentVolumeClaim:
        claimName: dynamic-pvc
```

Apply it:

```
bash
---------
kubectl apply -f pod-dynamic.yaml
kubectl get pod
```

---

## 5️⃣ Test the volume

```
bash
---------
kubectl exec -it dynamic-pv-pod -- sh
echo "Dynamic PV Test" > /data/test.txt
cat /data/test.txt
```

You should see: `Dynamic PV Test`

---

## 🧹 Cleanup

```
bash
---------
kind delete cluster --name dynamic-pv-demo
```

```
rm -rf /tmp/dynamic-pv
```

---

## 🧠 Summary

| Component | Description |
|---|---|
| StorageClass | Defines how storage is provisioned automatically |
| PVC | A user's request for storage |
| Local-path-provisioner | Provisions hostPath volumes dynamically |
| Pod | Uses the PVC to get access to persistent storage |

✅ **Yes, technically it exists on the host filesystem**, but: 🔒 **No, you won't see it directly unless you know exactly where to look and how to access it.**

---

## 💡 What's happening behind the scenes?

When you use:

```
bash
CopyEdit
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml
```

You're installing **local-path-provisioner**, which dynamically provisions volumes by writing to paths on the **node's local file system** (in this case, the Docker container running the control-plane in `kind`).

---

## 👀 So where is the file?

In `kind` (Kubernetes IN Docker), your cluster runs inside Docker containers. Each node (including the control-plane) is a Docker container.

The local-path-provisioner writes data to the default path:

```
lua
CopyEdit
/opt/local-path-provisioner/
```

inside the control-plane Docker container.

So when you run:

```
bash
CopyEdit
echo "Dynamic PV Test" > /data/test.txt
```

inside the pod, that file ends up (physically) somewhere under:

```
lua
CopyEdit
/opt/local-path-provisioner/pvc-xxxxx/
```

**inside the Docker container** that is running the node.

---

## 🔎 How to check manually?

1. List the Docker containers to find your `kind` control-plane:

```
bash
CopyEdit
docker ps
```

2. Get a shell into the container:

```
bash
CopyEdit
docker exec -it kind-control-plane sh
```

3. Browse to the storage path:

```
bash
CopyEdit
find /opt/local-path-provisioner -name test.txt
```

You should see your `test.txt` file there.

**What does "dynamic provisioning" mean in Kubernetes?**

A) Manually creating PersistentVolumes (PVs) before they are used
B) Automatically creating PersistentVolumeClaims (PVCs) from user input
C) Automatically creating PersistentVolumes (PVs) when a PVC is created using a StorageClass
D) Creating volumes only during cluster creation

✅ **Correct Answer:** C) Automatically creating PersistentVolumes (PVs) when a PVC is created using a StorageClass

---

**Which Kubernetes object is responsible for defining how dynamic storage is provisioned?**

A) Pod
B) Deployment
C) StorageClass
D) ConfigMap

✅ **Correct Answer:** C) StorageClass

---

**When using dynamic provisioning, what is the result of applying a PVC that references a valid StorageClass?**

A) Kubernetes waits for a matching PV to appear
B) Kubernetes creates a new PV automatically
C) Kubernetes throws an error
D) Kubernetes deletes existing PVs

✅ **Correct Answer:** B) Kubernetes creates a new PV automatically