**Kubernetes Storage Architecture: Volumes, PersistentVolumeClaims (PVCs), and VolumeClaimTemplates (VCs) in kubeadm**

**1. Overview**

Modern cloud-native applications often require persistent storage that survives container and pod restarts. Kubernetes, being a container orchestration platform, provides a robust storage framework to address such needs. This document explains the foundational concepts of **Volumes**, **PersistentVolumes (PVs)**, **PersistentVolumeClaims (PVCs)**, and **VolumeClaimTemplates (VCs)**, specifically in clusters set up using **kubeadm**.

**2. Kubernetes Volumes**

**2.1 Definition**

A **Volume** in Kubernetes is a directory, possibly backed by storage media, which is accessible to the containers in a pod. Kubernetes volumes extend the lifecycle of storage beyond container restarts within a pod, unlike Docker volumes which are tied to a single container instance.

**2.2 Characteristics**

* Volumes are mounted into containers at a specified path.
* A volume’s lifecycle is bound to the pod; it exists as long as the pod exists.
* Data is preserved across container crashes but not across pod deletions (except with PVs).

**2.3 Common Volume Types**

| **Type** | **Description** |
| --- | --- |
| *emptyDir* | Temporary storage shared across containers in a pod. Deleted with the pod. |
| *hostPath* | Mounts a path from the host node. Limited to development/test environments. |
| *configMap* | Injects configuration files into a pod. |
| *secret* | Provides sensitive information like passwords or tokens. |
| *nfs* | Mounts a network file system into the pod. |
| *csi* | Allows integration with external storage drivers via the CSI specification. |

**3. PersistentVolumes (PV)**

**3.1 Definition**

A **PersistentVolume** is a cluster-scoped resource that represents a piece of storage provisioned by an administrator or dynamically by Kubernetes through a **StorageClass**. It is independent of any individual pod and persists across pod lifecycles.

**3.2 Features**

* Pre-provisioned by an admin (static) or created on-demand (dynamic).
* Defined using a YAML configuration with capacity, access modes, and storage source.
* Managed by the control plane component kube-controller-manager.

**3.3 Access Modes**

| **Mode** | **Description** |
| --- | --- |
| ReadWriteOnce | Mounted as read-write by a single node. |
| ReadOnlyMany | Mounted as read-only by many nodes. |
| ReadWriteMany | Mounted as read-write by many nodes. Supported by some CSI drivers. |

**4. PersistentVolumeClaims (PVC)**

**4.1 Definition**

A **PersistentVolumeClaim** is a request for storage by a pod. It is a user-facing resource used to abstract away the details of how storage is provisioned and used.

**4.2 Lifecycle**

1. A PVC is created by a user/application.
2. The control plane searches for a matching PV that satisfies the claim.
3. Once bound, the PVC and PV remain bound for the life of the claim.

**4.3 PVC YAML Example**

*apiVersion: v1*

*kind: PersistentVolumeClaim*

*metadata:*

*name: example-pvc*

*spec:*

*accessModes:*

*- ReadWriteOnce*

*resources:*

*requests:*

*storage: 1Gi*

*storageClassName: standard*

**5. VolumeClaimTemplates (VCs)**

**5.1 Purpose**

**VolumeClaimTemplates** are used primarily with **StatefulSets**. Each replica of a StatefulSet gets a uniquely bound PVC created from the template. This ensures persistent, identity-aware storage across pod rescheduling.

**5.2 Key Benefits**

* Unique storage per pod.
* Automated PVC creation per replica.
* Essential for applications like databases, queues, and stateful services.

**5.3 Example in StatefulSet**

*volumeClaimTemplates:*

*- metadata:*

*name: data*

*spec:*

*accessModes: ["ReadWriteOnce"]*

*resources:*

*requests:*

*storage: 5Gi*

*storageClassName: standard*

**6. Dynamic Provisioning and StorageClasses**

**6.1 What is Dynamic Provisioning?**

Dynamic provisioning allows Kubernetes to automatically create storage volumes based on a PVC. This requires a **StorageClass** which defines how the storage should be provisioned (e.g., through AWS EBS, GCE PD, or a CSI driver).

**6.2 Example StorageClass**

*apiVersion: storage.k8s.io/v1*

*kind: StorageClass*

*metadata:*

*name: standard*

*provisioner: kubernetes.io/aws-ebs*

*parameters:*

*type: gp2*

**6.3 Important Considerations**

* kubeadm clusters do not come with a default StorageClass.
* You must deploy a CSI driver (like hostpath CSI for local testing or AWS EBS CSI for production).
* Persistent storage in production should always use a managed and replicated storage backend.

**7. kubeadm Considerations**

**7.1 Default kubeadm Setup**

* No dynamic provisioner is deployed by default.
* You may need to install a CSI driver manually.
* Ideal for bare-metal or custom environments where you want full control.

**7.2 Recommended CSI Drivers**

| **Environment** | **CSI Driver** |
| --- | --- |
| Local testing | hostpath CSI driver |
| AWS | aws-ebs-csi-driver |
| GCP | gcp-pd-csi-driver |
| Azure | azure-disk-csi-driver |
| On-prem storage | Ceph, OpenEBS, Longhorn, NFS CSI |

**8. Best Practices**

* Always use PVCs to request storage in pods, not PVs directly.
* Use StorageClass for dynamic provisioning instead of managing PVs manually.
* In production, ensure high availability and backup of persistent volumes.
* Avoid hostPath volumes in production due to node-dependence and security concerns.
* Use VolumeClaimTemplates with StatefulSets for applications requiring unique, stable storage.

**9. Conclusion**

Kubernetes provides a robust and extensible framework for managing storage through Volumes, PVs, PVCs, and VCs. When using **kubeadm**, administrators must ensure proper setup of storage provisioners and CSI drivers. Understanding these storage constructs is crucial for deploying and maintaining stateful applications effectively in a Kubernetes cluster.