Kubernetes Storage 101: Concepts and Best Practices

What is Kubernetes Storage?

Kubernetes is a free and open-source container orchestration platform. It provides services and management capabilities needed to efficiently deploy, operate, and scale containers in a cloud or cluster environment.

When managing containerized environments, Kubernetes storage is useful for storage administrators, because it allows them to maintain multiple forms of persistent and non-persistent data in a Kubernetes cluster. This makes it possible to create dynamic storage resources that can serve different types of applications.

If properly managed, the Kubernetes storage framework can be used to automatically provision the most appropriate storage to multiple applications, with minimal administrative overhead.

**In this article, you will learn:**

* [Kubernetes Storage Concepts](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#kubernetes-storage-concepts)
  + [Container Storage Interface (CSI)](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#container-storage-interface)
  + [Volumes](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#volumes)
  + [Non-Persistent Storage](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#non-persistent-storage)
  + [Persistent Volumes (PV) and Persistent Volume Claims (PVC)](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#persistent-volumes-and-persistent-volume-claims)
  + [StorageClasses](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#storageclasses)
  + [Dynamic Provisioning of StorageClasses](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#dynamic-provisioning-of-storageclasses)
* [Kubernetes Storage Best Practices](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#kubernetes-storage-best-practices)
  + [Kubernetes Volumes Settings](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#kubernetes-volumes-settings)
  + [Limiting Storage Resource Consumption](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#limiting-storage-resource-consumption)
  + [Resource Requests and Limits](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#resource-requests-and-limits)
* [Kubernetes Storage with Cloudian](https://cloudian.com/guides/understanding-kubernetes-multi-tenancy/kubernetes-storage-101-concepts-and-best-practices/#kubernetes-storage-with-cloudian)

**Kubernetes Storage Concepts**

Containers use the principle of immutability. This means that when containers are destroyed, including all data created during the lifetime of the containers. Immutability is not always appropriate, however. For example, applications that need to share information or maintain state cannot lose this information. In these cases, containers must have a location to permanently store information, which should outlive the lifespan of an individual container.

In Kubernetes, the most basic type of storage is non-persistent—also known as ephemeral. Each container has ephemeral storage by default—this storage uses a temporary directory on the machine that hosts the Kubernetes pod. It is portable, but not durable.

Kubernetes supports multiple types of persistent storage. This can include file, block, or object storage services from cloud providers (such as Amazon S3), storage devices in the local data center, or data services like databases. Kubernetes provides mechanisms to abstract this storage for applications running on containers, so that applications never communicate directly with storage media.

***Related content: read our guide to***[***hybrid cloud architecture***](https://cloudian.com/guides/hybrid-it/hybrid-cloud-architecture/)

**Container Storage Interface (CSI)**

CSI is a Kubernetes extension that simplifies storage management. Before CSI, users needed to integrate the data store’s device driver with Kubernetes, which was quite complex. CSI provides an extensible plugin architecture, so you can easily add plugins that support the storage devices and services used in your organization.

**Volumes**

Volumes are basic entities in Kubernetes, used to provide storage to containers. A volume can support all types of storage, including network file system (NFS), local storage devices, and cloud-based storage services. You can also build your own storage plugins to support new storage systems. Access to volumes can be achieved directly via pods or through persistent volumes (explained below).

**Non-Persistent Storage**

The default storage configuration in Kubernetes is non-persistent (temporary). As long as a container exists, it stores data in the temporary storage directory of the host, and when it shuts down the data is removed.

**Persistent Volumes (PV) and Persistent Volume Claims (PVC)**

To enable persistent storage, Kubernetes uses two key concepts:

**PersistentVolume (PV)** is a storage element in a cluster, defined manually by an administrator or dynamically defined by a storage class (explained below). A PV has its own lifecycle, separate from the lifecycle of Kubernetes pods. The PV API captures storage implementation details for NFS, cloud provider-specific storage systems, or iSCSI.

**PersistentVolumeClaim (PVC)** is a user’s storage request. An application running on a container can request a certain type of storage. For example, a container can specify the size of storage it needs or the way it needs to access the data (read only, write, read/write, with one-time or ongoing access).

Beyond storage size and access mode, administrators can offer PVs with custom properties, such as the type of disk (HDD vs. SSD), the level of performance, or the storage tier (regular or cold storage). Users can request storage based on these custom parameters without knowing the implementation details of the underlying storage. This is achieved using the StorageClass resource.

**StorageClasses**

You can configure **StorageClass** and assign PVs to each one. A StorageClass represents one type of storage. For example, one StorageClass may represent fast SSD storage, while another can represent magnetic drives, or remote cloud storage. This allows Kubernetes clusters to configure various types of storage according to workload requirements.

A StorageClass is a Kubernetes API for setting storage parameters using dynamic configuration to enable administrators to create new volumes as needed. The StorageClass defines the volume plug-in, external provider (if applicable), and the name of the container storage interface (CSI) driver that will enable containers to interact with the storage device.

**Dynamic Provisioning of StorageClasses**

Kubernetes supports dynamic volume configuration, allowing you to create storage volumes on demand. Therefore, administrators do not have to manually create new storage volumes and then create a PersistentVolume object for use in the cluster. When the user requests a specific type of storage, the entire process runs automatically.

The cluster administrator defines storage class objects as needed. Each StorageClass refers to a volume plugin, also called a **provisioner**. When a user creates a PVC, the provisioner automatically provisions a volume according to the required storage criteria.

**Kubernetes Storage Best Practices**

**Kubernetes Volumes Settings**

The Persistent Volume (PV) life cycle is independent of any particular container in the cluster. Persistent Volume Claims (PVC) are a request made by a container user or application for a specific type of storage.

When creating a PV, Kubernetes documentation recommends the following:

* Always include PVCs in the container configuration.
* Never include PVs in container configuration—because this will tightly couple a container to a specific volume.
* Always have a default StorageClass, otherwise PVCs that don’t specify a specific class will fail.
* Give StorageClasses meaningful names.

**Limiting Storage Resource Consumption**

It is advised to place limits on container usage of storage, to reflect the amount of storage actually available in the local data center, or the budget available for cloud storage resources.

There are two main ways to limit storage consumption by containers:

* **Resource Quotas**—limits the amount of resources, including storage, CPU and memory, that can be used by all containers within a Kubernetes namespace.
* **StorageClasses**—a StorageClass can limit the amount of storage provisioned to containers in response to a PVC.

**Resource Requests and Limits**

Kubernetes provides resource requests and resource limits that help you manage resource consumption allowed for individual containers.

A resource limit can be specified for temporary storage. Setting resource requests and limits can help prevent containers from being constrained by resource scarcity on container hosts, or taking up too many resources unexpectedly.

Use this command to ensure that all containers in a pod have resource requests and limits:

kubectl describe pod -n {your\_namespace} {your\_pod}