Kubernetes Essential

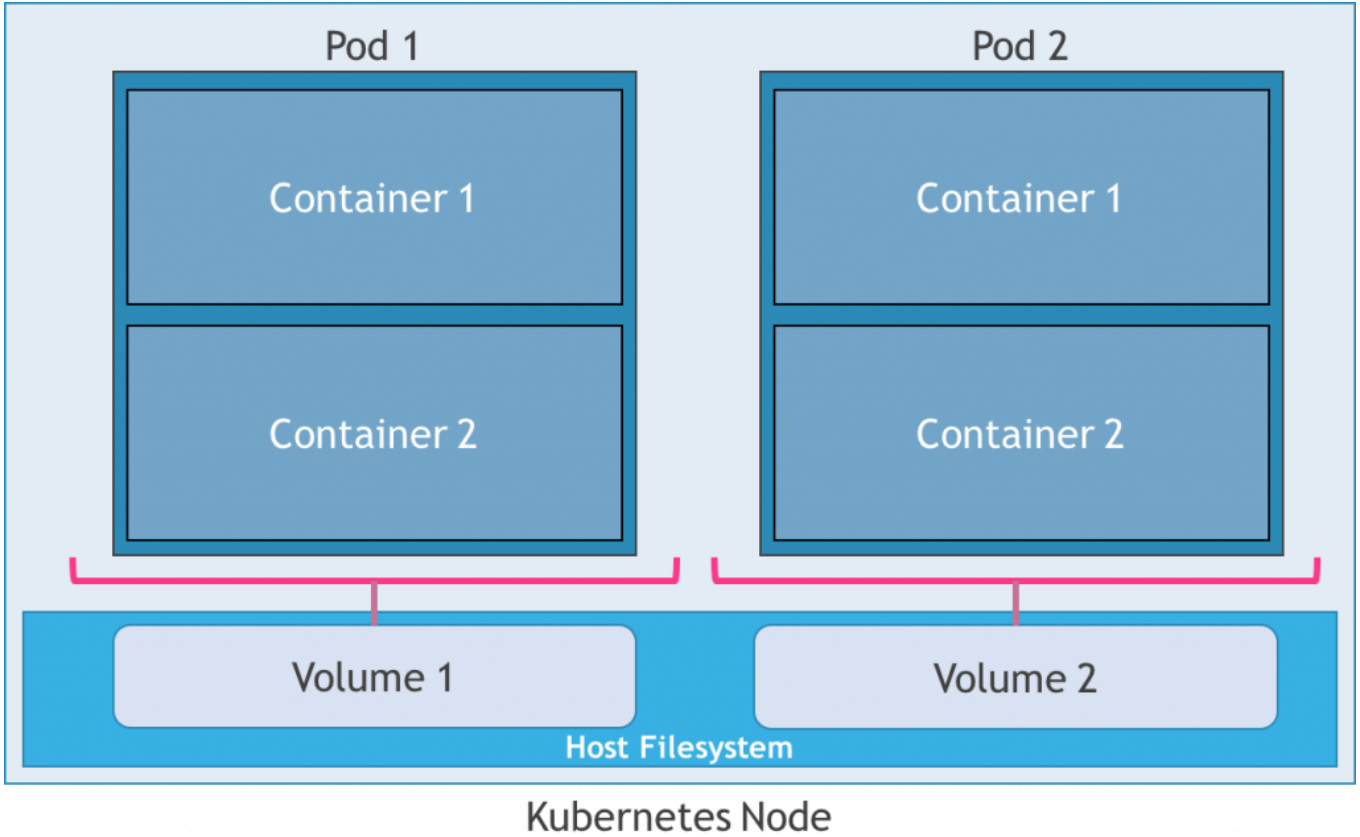
Agenda

- Assignment Review & Guides
- Kubernetes volume management
- Configmap and secret

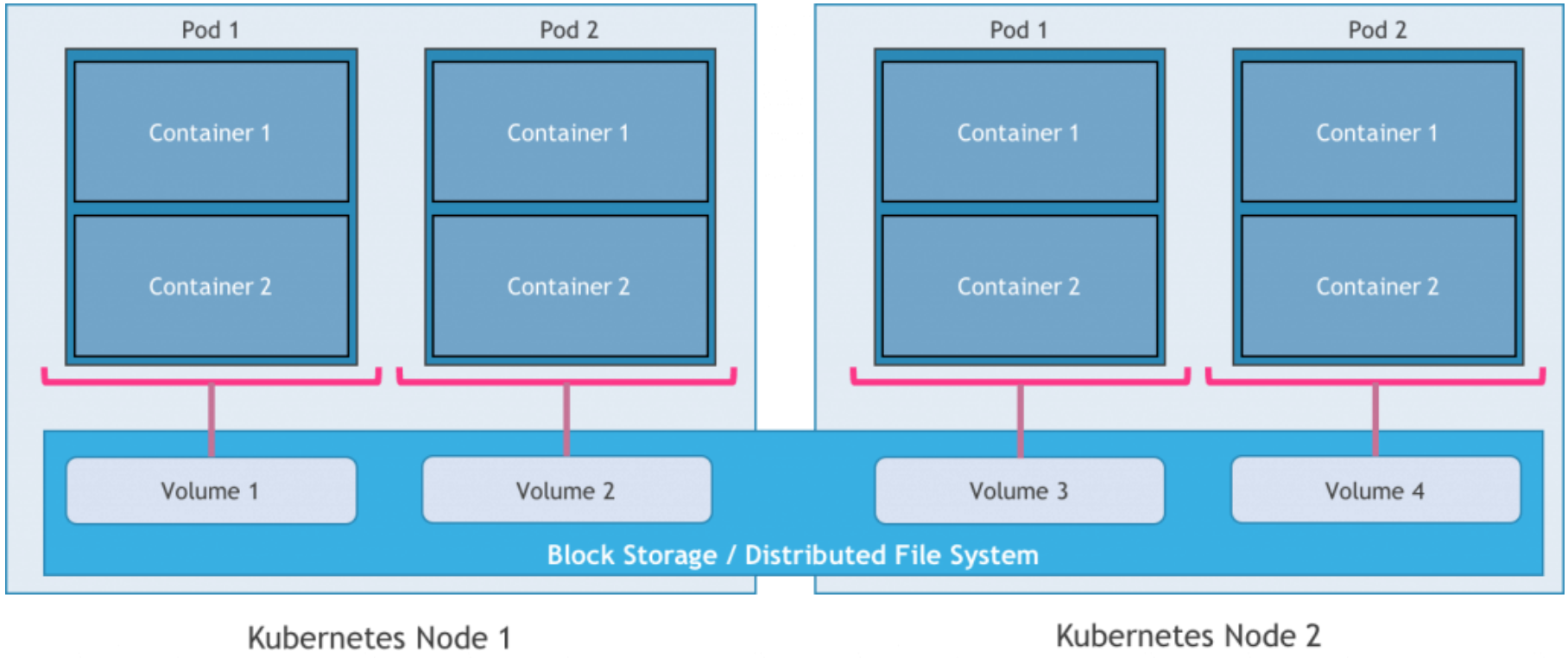
Persistent in Pods

- Pod are ephemeral and stateless
- Volumes bring persistence to Pods
- Kubernetes volumes a similar to Docker volumes, but managed differently
- All container in a Pod can access the volume
- Volumes are associated with the lifecycle of Pod
- Directory in the host are exposed as volumes
- Volumes may be based on a variety of storage backend
- A pod can have one or more types of volumes attached to it

Pod and Volumes



Pod and Volumes



Type of Volumes

- **Host-based**
  - emptyDir
  - hostPath
- **Block Storage**
  - awsElasticBlockStore
  - azureDisk
  - gcePersistentDisk
  - vsphereVolume
  - ...
- **Distributed File System**
  - NFS
  - Ceph
  - Gluster
  - Amazon EFS
  - Azure File System
  - ...
- **Other**
  - Flocker
  - iScsi

AWS EBS configuration example

If the EBS volume is partitioned, you can supply the optional field **partition: "<partition number>"** to specify which parition to mount on

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-ebs
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-ebs
      name: test-volume
  volumes:
  - name: test-volume
    # This AWS EBS volume must already exist.
    awsElasticBlockStore:
      volumeID: "<volume id>"
      fsType: ext4
```

emptyDir configuration example

An emptyDir volume is first created when a Pod is assigned to a node, and exists as long as that Pod is running on that node. As the name says, the emptyDir volume is initially empty

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

hostPath configuration example

A hostPath volume mounts a file or directory from the host node's filesystem into your Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: Directory
```

Persistent volumes

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.

It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV

Persistent volumes

- A **PersistentVolume** (PV) represents a storage resource.
- PVs are a **cluster wide resource** linked to a backing storage provider: NFS, EBS, GCEPersistentDisk, etc.
- Generally provisioned by an administrator.
- Their lifecycle is handled independently from a pod
- **CANNOT** be attached to a Pod directly. Relies on a **PersistentVolumeClaim** (PVC).
- A PVC is a **namespaced** request for storage.

Persistent volumes claim

A PersistentVolumeClaim (PVC) is a request for storage by a user.

It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory).

Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany).

Provision persistent volume and claim

There are two ways PVs may be provisioned: statically or dynamically

- **Static**

  A cluster administrator creates a number of PVs. They carry the details of the real storage, which is available for use by cluster users. They exist in the Kubernetes API and are available for consumption.

- **Dynamic**

  When none of the static PVs the administrator created match a user's PersistentVolumeClaim, the cluster may try to dynamically provision a volume specially for the PVC. This provisioning is based on StorageClasses

Provision persistent example

Each PV contains a spec and status, which is the specification and status of the volume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Volume Mode

Kubernetes supports two volumeModes of PersistentVolumes: **Filesystem and Block**

A volume with volumeMode: **Filesystem** is mounted into Pods into a directory. If the volume is backed by a block device and the device is empty, Kubernetes creates a filesystem on the device before mounting it for the first time.

You can set the value of volumeMode to **Block** to use a volume as a raw block device. Such volume is presented into a Pod as a block device, without any filesystem on it. This mode is useful to provide a Pod the fastest possible way to access a volume, without any filesystem layer between the Pod and the volume

Access Mode

The access modes are:

- ReadWriteOnce -- the volume can be mounted as read-write by a single node
- ReadOnlyMany -- the volume can be mounted read-only by many nodes
- ReadWriteMany -- the volume can be mounted as read-write by many nodes
- ReadWriteOncePod -- the volume can be mounted as read-write by a single Pod. This is only supported for CSI volumes and Kubernetes version 1.22+.

Storage class

- A StorageClass provides a way for administrators to describe the "classes" of storage they offer.
- Satisfies a set of requirements instead of mapping to a storage resource directly.
- Ensures that an application's claim for storage is portable across numerous backends or providers.
- Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators

The StorageClass Resource

- Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned.

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

The StorageClass Provisioner

Volume Plugin	Internal Provisioner	Config Example
AWSElasticBlockStore	✓	AWS EBS
AzureFile	✓	Azure File
AzureDisk	✓	Azure Disk
CephFS	-	-
Cinder	✓	OpenStack Cinder
FC	-	-
FlexVolume	-	-
Flocker	✓	-

The StorageClass Provisioner

GCEPersistentDisk	✓	GCE PD
Glusterfs	✓	Glusterfs
iSCSI	-	-
Quobyte	✓	Quobyte
NFS	-	NFS
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
ScaleIO	✓	ScaleIO
StorageOS	✓	StorageOS
Local	-	Local

CONFIGURATION

ConfigMaps

- An API object used to store non-confidential data in key-value pairs
- Externalized data stored within kubernetes
- Can be referenced through several different means:
  - ❑ environment variable
  - ❑ a command line argument (via environment variable)
  - ❑ injected as a configuration file into a volume mount
- Can be created from a manifest, literals, directories, or files directly.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  state: Minnesota
  city: Minneapolis
  content: |
    Look at this,
    its multiline!
```

Secrets

- Functionally identical to a ConfigMap.
- Stored as base64 encoded content.
- Encrypted at rest within etcd (if configured!).
- Ideal for username/passwords, certificates or other sensitive information that should not be stored in a container.
- Can be created from a manifest, literals, directories, or from files directly.

- **type**: There are three different types of secrets within Kubernetes:
  - **docker-registry** - credentials used to authenticate to a container registry
  - **generic/Opaque** - literal values from different sources
  - **tls** - a certificate based secret
- **data**: Contains key-value pairs of base64 encoded content.

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-secret
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password: bXlwYXNzd29yZA==
```

Q&A